



高等学校计算机教材

JSP

编程教程

◎ 郑阿奇 主编
◎ 周怡君 肖春兵 编著



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

<http://www.phei.com.cn>

高等学校计算机教材

JSP 编程教程

郑阿奇 主编

周怡君 肖春兵 编著

电子工业出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书包含 JSP 基础与入门、前端页面开发技术、服务器对象应用、JavaBean、数据库开发和综合应用等。读者入门比较容易,采用直接文本编辑和 MyEclipse 9.0 集成环境两种开发方式,每章上机练习与教程配套和同步,先引导编程,后操作练习。最后采用 Struts 框架对 JSP 进行综合应用训练。通过本教程的学习、上机练习和对综合应用实例的模仿,读者基本能够掌握用 JSP 解决小的应用问题。

本书可作为大学本科、高职高专有关课程的教材,也可作为 JSP 培训用书及计算机用户的参考用书。本书配有免费的教学课件、解密开发环境包、实例文件、综合应用源文件,可在华信教育资源网(网址为 www.hxedu.com.cn)免费下载。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

JSP 编程教程 / 郑阿奇主编; 周怡君, 肖春兵编著. —北京: 电子工业出版社, 2012. 8

高等学校计算机教材

ISBN 978-7-121-17828-3

I. ①J… II. ①郑… ②周… ③肖… III. ①JAVA 语言—网页制作工具—程序设计—高等学校—教材
IV. ①TP312②TP393.092

中国版本图书馆 CIP 数据核字(2012)第 181408 号

策划编辑: 郝黎明

责任编辑: 郝黎明 特约编辑: 张 彬

印 刷:

装 订:

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1092 1/16 印张: 21.75 字数: 556.8 千字

印 次: 2012 年 8 月第 1 次印刷

印 数: 4 000 册 定价: 39.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线: (010) 88258888。

前 言

JSP 作为当前最热门的网页开发技术，越来越受到人们的普遍关注。许多高等学校的多个专业已开设了 JSP 课程，我们在总结 JSP 开发技术、汲取实用教程教材编写经验后编写本书。

本书主要内容包括 JSP 基础与入门、前端页面开发技术、服务器对象应用、JavaBean、数据库开发和综合应用等。基础与入门介绍网页设计基础、工作原理、JSP 环境、JSP 编程基础；前端页面开发技术介绍 JavaScript 和 Ajax；服务器对象应用介绍 9 种基本服务器对象及其应用；JavaBean 介绍结构属性、作用域、编程开发和第三方组件的应用；数据库开发以 SQL Server 2008 为主介绍数据库的基本概念、安装配置、与 MyEclipse 9.0 的集成、JSP 对数据库的基本操作，同时介绍通过 JDBC-ODBC 桥连接 Access 2007 和 Excel 2007，以及用 JSP 对其进行简单操作；综合应用以 Struts 为框架，综合应用各种技术解决一个实际问题，功能虽然简单，但可体现 JSP 的主要内容。

本书主要有下列特点。

- (1) 读者入门比较容易。JSP 入门一章介绍 JSP 的原理，理清开发思路。
- (2) 全书灵活应用直接文本编辑和 MyEclipse 9.0 集成环境开发方式，有利于学习和应用开发。
- (3) “上机练习”部分与教程配套和同步，通过实例先引导编程，然后提出问题请读者思考，并请读者自己进行修改和扩充练习；“综合应用”部分采用 Struts 框架对 JSP 进行综合应用训练。
- (4) 突出前端开发和后端开发，并且配合使用，有利于读者掌握系统开发。

本书不仅适合教学，也非常适合用 JSP 编程和开发应用程序的用户学习和参考。通过对本书的学习、对实验操作和综合应用实例的模仿，读者基本能够掌握怎样用 JSP 解决小的应用问题。

本书由南京师范大学郑阿奇主编，东南大学周怡君、南京师范大学肖春兵编著。参加本套丛书编写的还有梁敬东、顾韵华、王洪元、刘启芬、丁有和、曹弋、徐文胜、殷红先、张为民、姜乃松、彭作民、高茜、陈冬霞、钱晓军、朱毅华、时跃华、周何骏、赵青松、周淑琴、陈金辉、李含光、王一莉、徐斌、王志瑞、孙德荣、周怡明、刘博宇、郑进、刘毅、陈杰、刘有春等。

本书配有教学课件、解密开发环境包、实例文件和综合应用源文件，需要者可在华信教育资源网（网址为 www.hxedu.com.cn）免费下载。

由于作者水平有限，不当之处在所难免，恳请读者批评指正。

编 者
2012.7

目 录

第 1 章 Web 网页设计基础	1
1.1 Web 简介	1
1.1.1 Web 的概念	1
1.1.2 Web 工作原理	2
1.1.3 WWW 服务	2
1.2 XHTML 语言	3
1.2.1 XHTML 文档基本构成	3
1.2.2 XHTML 格式标记	7
1.2.3 XHTML 多媒体标记	13
1.2.4 XHTML 基本应用	16
1.2.5 框架网页设计	28
1.3 CSS 初步	31
1.3.1 CSS 定义及引用	31
1.3.2 CSS 选择符	34
1.3.3 CSS 属性	36
1.4 动态网页	40
1.4.1 何谓“动态”网页	40
1.4.2 动态网站架构原理	41
1.4.3 Web 开发工具	41
1.5 上机练习	42
第 2 章 JSP 入门	45
2.1 安装 JSP 运行环境	45
2.1.1 JDK 7 的安装与配置	45
2.1.2 Tomcat 7 的安装与配置	46
2.1.3 MyEclipse 9.0 介绍	48
2.2 JSP 软件工作原理	50
2.2.1 Servlet 基础	50
2.2.2 JSP 软件体系结构	61
2.2.3 JSP 程序执行流程	63
2.3 一个简单的 JSP 例子	63
2.3.1 JSP 实现圆面积计算	63
2.3.2 JSP 程序基本构成	65
2.3.3 JSP 运行机制分析	67
2.4 JSP+JavaBean 结构程序	70

2.5	MyEclipse 开发 JSP 程序	73
2.5.1	配置 JRE	73
2.5.2	集成 MyEclipse 与 Tomcat	74
2.5.3	MyEclipse 开发入门	76
2.6	上机练习	81
第 3 章	JSP 基础编程	82
3.1	Java 基础	82
3.1.1	数据类型、运算符和表达式	82
3.1.2	条件、循环语句	88
3.1.3	自定义函数、变量声明	94
3.1.4	数组	95
3.1.5	面向对象程序设计	96
3.2	JSP 系统常用类	100
3.2.1	常用数值类	100
3.2.2	常用字符串类	106
3.2.3	常用日期/时间类	112
3.2.4	常用系统信息类	114
3.3	JSP 编译指令	119
3.3.1	include 指令	119
3.3.2	page 指令	121
3.3.3	taglib 指令	122
3.4	JSP 动作元素	122
3.4.1	<jsp:param>	122
3.4.2	<jsp:include>	122
3.4.3	<jsp:useBean>	124
3.4.4	<jsp:setProperty>与<jsp:getProperty>	126
3.4.5	<jsp:forward>	128
3.4.6	<jsp:plugin>	128
3.5	上机练习	130
第 4 章	前端页面开发技术	131
4.1	JavaScript 基础	131
4.1.1	脚本语言简介	131
4.1.2	网页中的 JavaScript	132
4.1.3	基本语法	132
4.1.4	常用语句	135
4.1.5	对象	136
4.1.6	事件	137

4.2	JavaScript 浏览器对象	142
4.2.1	浏览器对象的概念	142
4.2.2	Window 对象	142
4.2.3	Document 对象	145
4.2.4	History 对象	147
4.2.5	Navigator 对象	147
4.2.6	Location 对象	148
4.2.7	Link 对象	148
4.3	JavaScript 页面开发实例	148
4.3.1	制作隐式菜单	148
4.3.2	青奥会倒计时牌	151
4.3.3	图像自由运动	153
4.4	Ajax 技术	157
4.4.1	Ajax 的概念	157
4.4.2	Ajax 基础	158
4.4.3	Ajax 应用实例	161
4.5	上机练习	166
第 5 章	JSP 服务器对象应用	168
5.1	内置对象及其作用	168
5.1.1	JSP 内置对象	168
5.1.2	客户端/服务器交互	169
5.1.3	对通信的控制	169
5.2	请求对象: Request	170
5.2.1	获取请求参数	170
5.2.2	设置/获取属性	172
5.2.3	获取其他信息	173
5.2.4	Request 方法一览	175
5.3	响应对象: Response	175
5.3.1	发送 HTTP 文件头	176
5.3.2	页面重定向	179
5.3.3	缓冲区输出	179
5.3.4	Response 方法一览	181
5.4	会话对象: Session	181
5.4.1	Session 原理	181
5.4.2	数据存取	182
5.4.3	超时管理	184
5.4.4	Session 方法及应用	184
5.5	共享对象: Application	188

5.5.1	作用范围	188
5.5.2	全局网页应用	188
5.5.3	Application 方法一览	190
5.6	其他对象	191
5.6.1	Out 对象	191
5.6.2	Page 对象	193
5.6.3	PageContext 对象	193
5.6.4	Config 对象	195
5.6.5	Exception 对象	196
5.7	Cookie 及应用	200
5.7.1	创建 Cookie 对象	200
5.7.2	Cookie 对象的主要方法	200
5.7.3	Cookie 对象与 Session 对象的比较	201
5.7.4	示例	201
5.8	综合应用——简易留言板	203
5.9	上机练习	210
第 6 章	JavaBean 及其应用	212
6.1	JavaBean 简介	212
6.1.1	使用 JavaBean 的原因	212
6.1.2	JavaBean 的形式和要素	213
6.2	JavaBean 基本结构	214
6.2.1	JavaBean 的属性	214
6.2.2	JavaBean 的方法	218
6.2.3	JavaBean 的事件	218
6.3	JavaBean 的作用域	219
6.3.1	作用域	219
6.3.2	获取作用域数据	220
6.4	JavaBean 实现动态日历	222
6.5	第三方 JavaBean 组件的应用	227
6.5.1	文件上传	227
6.5.2	文件下载	231
6.6	上机练习	233
第 7 章	JSP 操作数据库	234
7.1	数据库基础	234
7.1.1	关系模型	234
7.1.2	SQL 语言	236
7.1.3	流行的 DBMS	237

7.1.4	JSP 数据访问模型	239
7.2	SQL Server 2008 基础	239
7.2.1	安装配置	240
7.2.2	SQL Server 2008 服务器组件	242
7.2.3	Management Studio 环境	242
7.2.4	建立数据库和表	244
7.3	JDBC 连接 SQL Server 数据库	246
7.3.1	在 MyEclipse 中创建连接	246
7.3.2	解决 Tomcat 与 SQL Server 2008 端口冲突	248
7.3.3	测试连接的可用性	249
7.4	JSP 操作 SQL Server 数据库	254
7.4.1	添加记录	255
7.4.2	查询记录	256
7.4.3	更新记录	270
7.4.4	删除记录	274
7.4.5	使用存储过程	277
7.5	与其他数据库的互操作	280
7.5.1	连接 Access 2007	280
7.5.2	连接 Excel 2007	287
7.6	上机练习	289
第 8 章	JSP 综合应用开发	291
8.1	JSP 系统的架构方式	291
8.1.1	表示层的两种架构模式	291
8.1.2	MVC 基础	292
8.1.3	Struts 1 框架	294
8.2	开发前的准备工作	295
8.2.1	创建 Web 工程	295
8.2.2	导入 Struts 库	297
8.2.3	数据准备	300
8.2.4	配置过滤器	301
8.3	模型组件的开发	304
8.3.1	连接数据库	304
8.3.2	定义 POJO 类	305
8.3.3	操作数据库	307
8.4	登录验证功能	308
8.4.1	登录页面设计	308
8.4.2	Action 功能模块	311
8.4.3	后台的验证操作	314

8.4.4	登录成功页	315
8.5	学生信息检索	316
8.5.1	检索页面显示	316
8.5.2	检索 Action 模块	320
8.5.3	后台数据检索操作	322
8.5.4	测试检索功能	324
8.6	新生信息录入和删除	326
8.6.1	新生录入页面	326
8.6.2	插入 Action 模块	329
8.6.3	后台数据插入操作	333
8.6.4	录入新生信息	333
8.6.5	删除学生信息	335
8.7	上机练习	337

Web 网页设计基础

在今天的社会生活中，Internet（因特网）无处不在，它集合了全球绝大多数重要的信息资源，是信息时代人们进行交流不可或缺的工具。WWW(World Wide Web)简称 Web，是 Internet 提供的一项最基本、应用最广泛的服务，而学习 Web 网页设计更成为了一种时尚。

1.1 Web 简介

1.1.1 Web 的概念

Web 是存储在 Internet 计算机中数量巨大的文档的集合。这些文档称为页面，是一种超文本（Hypertext）信息，可以用于描述超媒体。文本、图形、视频、音频等多媒体称为超媒体（Hypermedia）。Web 上的信息是由彼此关联的文档组成的，而使其连接在一起的便是超链接（Hyperlink）。

Web 页面就是我们在浏览器里看到的网页，它组织在一个文件中，文件的位置在浏览器的地址栏中采用 URL 规则指定。

1. 网页

网页一般用 HTML/XHTML 语言写成，在网页中可以嵌入文本、图形、音频和视频信息，是一种多媒体作品。HTML/XHTML 本身只能描述静态的 Web 页面，但在其中可以嵌入 Java、JavaScript、ActiveX、VBScript、VRML 等语言，以完成非常复杂的任务。

2. 主页（或首页）

主页可以认为是一组网页中最主要的网页，是进入其他网页的起始网页，主页通过超链接链接到其他网页。

3. 超链接

Web 上的信息是由彼此关联的文档组成的，而使其连接在一起的是超链接。超链接是 HTML 语言中的一个标记，标记中显示的内容较之其他内容有明显特征，如颜色不同、带有下画线等。标记中的一个属性的值指向链接到的另一个网页的 URL。在超链接标记中显示的内容位置单击鼠标，通过超链接即可转到指定的网页。

4. 网站

若干个网页按一定方式连接起来形成一个整体，用来描述一组完整的信息。这样一组存放

在 Web 服务器上具有共同主题的相关联的网页组成的一组资源称为网站。网站的网页总由一个主页和若干个其他网页组成。主页也可以认为是网站的门面。

1.1.2 Web 工作原理

从本质上讲, Web 是基于客户机/服务器 (C/S) 的一种体系结构, 一般用户的计算机称为客户机, 用于提供服务的机器称为服务器。在 Web 方式下, 客户端常用浏览器访问服务器, 客户机向服务器发送请求, 要求执行某项任务, 而服务器执行此项任务, 并向客户机返回响应, 如图 1.1 所示。Web 客户程序叫做浏览器 (Browser), 而浏览器程序基本上都是标准化的, 因此, Web 体系结构又可以称为浏览器/服务器 (B/S) 结构。

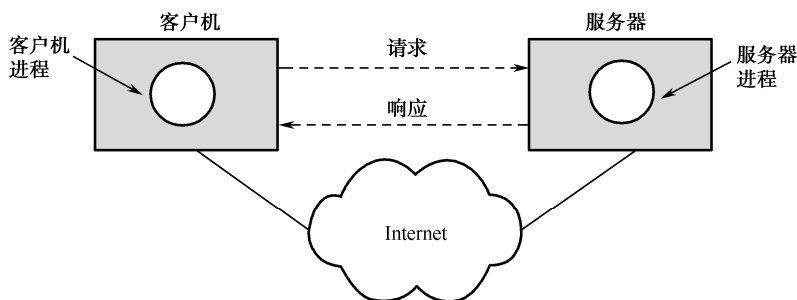


图 1.1 客户机/服务器结构

在客户机/服务器体系结构中, 通常很容易将客户机和服务器理解为两端的计算机。但事实上, “客户机”和“服务器”在概念上更多指软件, 即两台机器上相应的应用进程。

1.1.3 WWW 服务

Internet 包含成千上万的 WWW 服务器, 其上集中了全球的信息资源, 是存储和发布信息的地方, 也是人们查询信息的场所。

Web 浏览器和服务器用超文本传输协议 (HTTP 协议) 来传输 Web 文档, 通过统一资源定位符 URL 标识文档在网络上服务器的位置及服务器中的路径, 如图 1.2 所示。

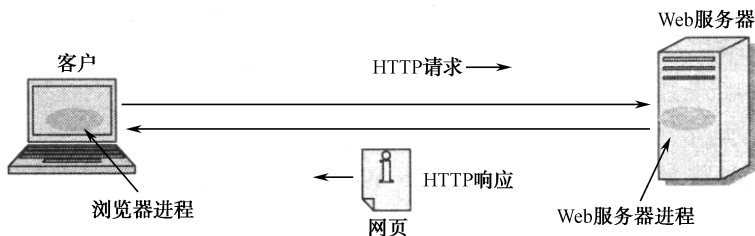


图 1.2 传输 Web 文档

Web 文档是用超文本标记语言 HTML (Hypertext Markup Language) 编制的文档文件, 由浏览器解释并显示在用户浏览器的窗口中。HTML 是一种简单、通用的标记语言, 可以制作包含文字、表格、图像、声音等精彩内容的网页, 目前流行的版本是 HTML 4.01, 最新版本为 HTML 5。XHTML 是可扩展超文本标签语言 (EXtensible HyperText Markup Language), XHTML

1.0 与 HTML 4.01 几乎是相同的, 是符合 W3C 标准的更严谨、更纯净的 HTML 版本。

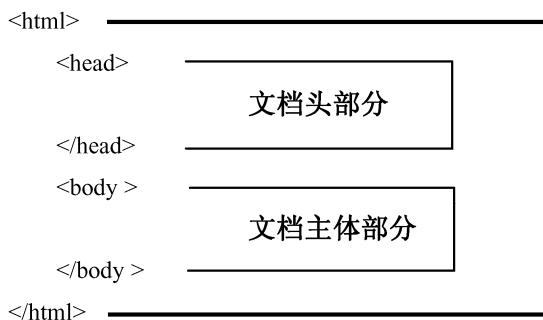
网页设计在客户端需要使用 XHTML 语言、CSS 样式表, 本章分别对它们加以介绍。

1.2 XHTML 语言

1.2.1 XHTML 文档基本构成

一个 XHTML 文档由 DOCTYPE、head 和 body 三个主要的部分构成, 基本的文档结构如下:

<!DOCTYPE 文档类型声明...>



在 XHTML 文档中, 文档类型声明总位于首行, 文档的其余部分类似于 HTML。基本的 HTML 页面从 `<html>` 标记开始, 以 `</html>` 标记结束, 其他所有 HTML 代码都位于这两个标记之间。 `<head>` 与 `</head>` 之间是文档头部分, `<body>` 与 `</body>` 之间是文档主体部分。

下面是一个简单的 (最小化的) XHTML 文档:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>simple document</title>
  </head>
  <body>
    <p>a simple paragraph</p>
  </body>
</html>
```

这种文档用普通的记事本就可以创建、编辑, 并且可以在各种操作系统平台 (如 UNIX、Windows 等) 中执行。

1. 文档类型声明

文档类型声明定义文档的类型, 包括三种文档类型。

(1) Strict (严格类型)

在此情况下, 需要干净的标记, 避免表现上的混乱, 与层叠样式表 (CSS) 配合使用。上面这个最小化的 XHTML 文档用的就是 Strict 类型:

```
<!DOCTYPE html
```

```
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

(2) Transitional（过渡类型）

当需要利用 HTML 在表现上的特性，为那些不支持 CSS 的浏览器编写 XHTML 时使用 Transitional，格式如下：

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

(3) Frameset（框架类型）

当需要使用 HTML 框架将浏览器窗口分割为两部分或更多框架时使用 Frameset，格式如下：

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

2. 文档头

文档头部分处于<head>与</head>标记之间，在文档头部分一般可以使用以下几种标记。

- <title>和</title>：指定网页的标题。例如，“<title>主页</title>”表示该网页的标题为“主页”，在浏览器标题栏中显示的文本即为“主页”，通常 Web 搜索工具用它作为索引。
- <style>和</style>：指定文档内容的样式表，如字体大小、格式等。在文档头部分定义了样式表后，就可以在文档主体部分引用样式表，具体用法详见 1.3 节。
- <!--和-->：用于注释内容，其之间的内容为 XHTML 的注释部分。
- <meta>：描述标记，用于描述网页文档的属性参数。

描述标记的格式为<meta 属性=“值” ... />，常用的属性有 name、content 和 http-equiv。name 为 meta 的名字；content 为页面的内容；http-equiv 为 content 属性的类别。http-equiv 取不同值时，content 表示的内容也不一样。

◆ http-equiv="Content-type"时，content 表示页面内容的类型，例如：

```
<meta name="description" http-equiv="Content-type" content="text/html; charset=gb2312" />
```

表示 meta 的名称为 description，网页是 XHTML 类型，编码规则是 gb2312。

◆ http-equiv="refresh"时，content 表示刷新页面的时间，例如：

```
<meta http-equiv="refresh" content="10; URL=xxx.htm" />
```

表示 10 秒后进入 xxx.htm 页面，如果不加 URL 则表示每 10 秒刷新一次本页面。

◆ http-equiv="Content-language"时，content 表示页面使用的语言，例如：

```
<meta http-equiv="Content-language" content="en-us" />
```

表示页面使用的语言是美国英语。

◆ http-equiv="pics-Label"时，content 表示页面内容的等级。

◆ http-equiv="expires"时，content 表示页面过期的日期。

● <script>和</script>：在这两个标记之间可以插入脚本语言程序，例如：

```
<script language="javascript">
    alert("你好！");
</script>
```

表示插入的是 JavaScript 脚本语言，脚本语言主要用于客户端（前端）页面开发，详细内容将在本书第 4 章介绍。

3. 文档主体

<body>和</body>是文档正文标记，文档的主体部分就处于这两个标记之间。<body>标记中还可以定义文档主体的一些内容，格式如下：

```
<body 属性="值"... 事件="执行的程序"...> ... </body>
```

<body>标记常用的属性如下。

● background：文档背景图片的 URL 地址。例如：

```
<body background="back-ground.gif">
```

表示文档背景图片名称为 back-ground.gif，上面的代码中没有给出图片所在的位置，则表示图片和文档文件在同一文件夹下，如果图片和文档文件不在同一位置，则需要给出图片的路径，例如：

```
<body background="C:/image/back-ground.gif">
```

说明：在指定文件位置时，为防止与转义符“\”混淆，一般用“/”来代替“\”。

● bgcolor：文档的背景颜色。例如：

```
<body bgcolor="red">
```

表示文档的背景颜色为红色。系统的许多标记都会用到颜色值，颜色值一般用颜色名称或十六进制数值来表示，表 1.1 列出了 16 种标准颜色的名称及其十六进制数值。

表 1.1 16 种标准颜色的名称及其十六进制数值

颜 色	名 称	十六进制数值	颜 色	名 称	十六进制数值
淡蓝	aqua(cyan)	#00FFFF	海蓝	navy	#000080
黑	black	#000000	橄榄色	olive	#808000
蓝	blue	#0000FF	紫	purple	#800080
紫红	fuchsia(magenta)	#FF00FF	红	red	#FF0000
灰	gray	#808080	银色	silver	#C0C0C0
绿	green	#008000	淡青	teal	#008080
浅绿	lime	#00FF00	白	white	#FFFFFF
褐红	maroon	#800000	黄	yellow	#FFFF00

● text：文档中文本的颜色，例如：

```
<body text="blue">
```

表示文档中文字的颜色都为蓝色。

● link：文档中链接的颜色。

● vlink：文档中已被访问过的链接的颜色。

● alink：文档中正在被选中的链接的颜色。

正文标记中的常用事件有 onload 和 onunload。onload 表示文档首次加载时调用的事件处理程序，onunload 表示文档卸载时调用的事件处理程序。

4. 示例

【例 1.1】使用 XHTML 设计一个以南京师范大学校门为背景的网页。

(1) 准备

在桌面上新建一个 XHTML 文件夹，到南京师范大学校园网下载一幅仙林新校区大门的照片，以文件名 njnu.jpg 保存于 XHTML 文件夹下。

(2) 编辑 XHTML 文档

打开 Windows 记事本，输入下列内容：

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
  <title>南京师范大学校门</title>
  <script language="JavaScript">
    function myp()
    {
      alert("欢迎访问!");
    }
  </script>
</head>
<body background="njnu.jpg" onload="myp()">
  <div align="center">
    <font color="#8888FF" size="7">
      南京师范大学仙林新校区
    </font>
  </div>
</body>
</html>
```

以 1_njnu.html 作为文件名保存到 XHTML 文件夹中（与 njnu.jpg 在同一个文件夹）。

(3) 运行

双击打开刚刚编辑的文档 1_njnu.html，将显示如图 1.3 所示的页面。



图 1.3 南京师范大学校门背景网页

从上例的代码中可以很清楚地看出 XHTML 与 HTML 有以下几点最主要的不同之处。

① 元素必须被正确地嵌套；

- ② XHTML 元素必须被关闭;
- ③ 标签名必须用小写字母;
- ④ XHTML 文档必须拥有一个根元素。

另外, XHTML 还有下列语法规则。

- ① 属性名称必须小写;
- ② 属性值必须加引号;
- ③ 属性不能简写;
- ④ 用 id 属性代替 name 属性;
- ⑤ XHTML DTD 定义了强制使用的 HTML 元素。

XHTML 页面中显示的内容都是在文档的主体部分(即<body>和</body>标记之间)定义的。文档主体部分能够定义文本、图像、声音、滚动字幕、表格、表单、超链接和框架等,这些都依靠丰富的 XHTML 语言标记来完成。

1.2.2 XHTML 格式标记

文本是网页的重要内容。编写 XHTML 文档时,可以将文本放在标记之间来设置文本的格式。文本格式包括分段、换行、段落对齐方式、字体、字号、文本颜色及字符样式等。

1. 分段标记

分段标记的格式如下:

```
<p 属性="值"...>...</p>
```

段落是文档的基本信息单位,利用分段标记可以忽略文档中原有的回车和换行来定义一个新段落,或换行并插入一个空格。

单独用<p>标记时会空一行,使后续内容隔行显示;同时使用<p>和</p>标记则将段落包围起来,表示一个分段的块。

分段标记常用属性为 align,表示段落的水平对齐方式。其取值可以是 left(左对齐)、center(居中)、right(右对齐)和 justify(两端对齐),例如:

```
<p align="center">分段标记演示</p>
```

在下面的标记中还会经常用到 align 属性,当该属性省略时取默认值 left。

2. 换行标记

换行标记为
,该标记将强行中断当前行,使后续内容在下一行显示。

3. 标题标记

标题标记的格式如下:

```
<h1 属性="值">...</h1>
```

其中 hn 的取值为 h1、h2、h3、h4、h5 和 h6,都表示黑体,h1 表示字最大,h6 表示字最小。标题标记的常用属性也是 align,与分段标记类似。

4. 对中标记

对中标记的格式如下:

```
<center>...</center>
```

此标记的作用是将标记中间的内容全部居中。

5. 块标记

块标记的格式如下:

```
<div 属性="值"...>...</div>
```

块标记的作用是定义文档块，常用的属性也是 align。

【例 1.2】应用前面提到的各种标记。

新建 1_2ex.html 文件，输入以下代码：

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
    <title>标记应用</title>
</head>
<body>
    <p align="center">分段标记</p>
    换行标记<br />
    <center>对中标记</center><br /><br />
    <div align="center">下面使用了 div 标记
        <h1>标题标记 1</h1>
        <h2>标题标记 2</h2>
        <h3 align="left">标题标记 3</h3>
    </div>
</body>
</html>
```

双击 1_2ex.html 文件，运行结果如图 1.4 所示。



图 1.4 1_2ex.html 运行结果

实际上，<div>标记在更多情况下用于布局，例如：

```
<div id="top" >
    <div id="logo"> ...</div>
    <div id="ad">...</div>
```

```
<div id="set">...</div>
</div>
<div id="center">
  <div id="left">...</div>
  <div id="right">...</div>
</div>
<div id="bottom">
</div>
```

如此设置样式，布局效果如图 1.5 所示。

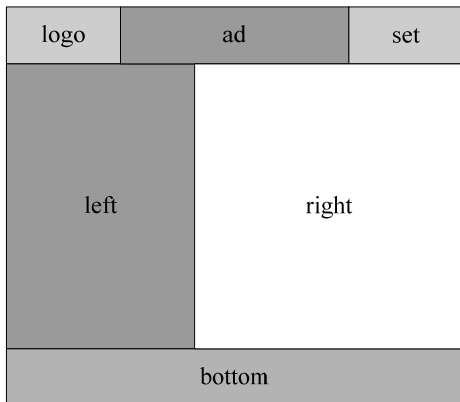


图 1.5 <div>的布局

另外，用于在一行内布局。它仅在行内定义一个区域，即在一行内可以被数个 span 元素划分成几个区域，从而实现某种特定的布局效果。不仅如此，span 元素还能定义宽和高，例如：

```
<div id=" top"... >
  <span ...> ... </span>
  <span ...> ...
  <span ...> ... </span>
</span>
</div>
```

span 元素作为文本或其他内联元素的容器，与 div 元素一样在 CSS 布局中有着不可忽视的作用。

6. 水平线标记

水平线标记用于在文档中添加一条水平线，分隔文档，格式如下：

```
<hr 属性="值"... />
```

该标记常用的属性有 align、color、noshade、size 和 width。align 表示水平线的对齐方式；color 表示线的颜色；noshade 没有值，显示一条无阴影的实线；size 是线的宽度（以像素为单位）；width 是线的长度（像素或百分比）。例如：

```
<hr />
<hr size="2" width="300" noshade ="noshade" />
<hr size="6" width="60%" color="red" align="center" />
```

7. 字体标记

字体标记用于设置文本的字符格式，主要包括字体、字号和颜色等，格式如下：


```
<font 属性="值"...>...</font>
```

该标记常用的属性如下。

- **face**: 其值为一个或多个字体名, 中间用逗号隔开。浏览器首先使用第 1 种字体显示标记内的文本。如果浏览器所在的计算机中没有安装第 1 种字体, 则尝试使用第 2 种字体……以此类推, 直到找到匹配的字体为止。如果 **face** 中列出的字体都不符合, 则使用默认字体。例如:

```
<font face="黑体,楷体-GB2312,仿宋-GB2312">设置字体</font>
```

- **size**: 指定字体的大小, 值为 1~7, 默认值为 3。size 值越大, 字就越大。也可以使用“+”或“-”来指定相对字号。例如:

```
<font size="6">这是 6 号字</font>
```

```
<font size="+3">这也是 6 号字</font>
```

- **color**: 指定字体的颜色, 颜色值在表 1.1 中已经列出。

8. 固定字体标记

固定字体标记的格式如下:

```
<b>粗体</b>
```

```
<i>斜体</i>
```

```
<big>大字体</big>
```

```
<small>小字体</small>
```

```
<tt>固定宽度字体</tt>
```

9. 标线标记

标线标记的格式如下:

```
<sup>上标</sup>
```

```
<sub>下标</sub>
```

```
<u>下划线</u>
```

```
<s>删除线</s>
```

10. 特殊标记

在网页中, 一些特殊符号 (如多个空格和版权符号“©”等) 是不能直接输入的, 这时可以使用字符实体名称或数字表示方式。例如, 要在网页中输入一个空格, 可以输入“ ”或“ ”。表 1.2 列出了一些常用的特殊符号和它们的实体名称及数字表示。

表 1.2 常用的特殊符号和它们的实体名称及数字表示

字 符	说 明	字符实体名称	数字表示	字 符	说 明	字符实体名称	数字表示
	无断行空格	 	 	¥	元符号	¥	¥
¢	美分符号	¢	¢	§	节符号	§	§
£	英镑符号	£	£	©	版权符号	©	©
®	注册符号	®	®	&	“and”符号	&	&
°	度	°	°	<	小于符号	<	<
²	平方符号	²	²	>	大于符号	>	>
³	立方符号	³	³	€	欧元符号	€	€

11. 列表标记

列表标记可以分为有序列表标记、无序列表标记和描述性列表标记。

(1) 有序列表标记

有序列表是在各列表项前面显示数字或字母的缩排列表,可以使用有序列表标记``和列表项标记``来创建,格式如下:

```
<ol 属性="值"...>
  <li>列表项 1</li>
  <li>列表项 2</li>
  ...
  <li>列表项 n</li>
</ol>
```

说明:

- ``标记用于控制有序列表的样式和起始值,它通常有 `start` 和 `type` 两个常用的属性。`start` 是数字序列的起始值;`type` 是数字序列的列样式,`type` 值有 `1`、`A`、`a`、`I`、`i`。`1` 表示阿拉伯数字 1、2、3 等;`A` 表示大写字母 A、B、C 等;`a` 表示小写字母 a、b、c 等;`I` 表示大写罗马数字 I、II、III 等;`i` 表示小写罗马数字 i、ii、iii 等。
- ``标记用于定义列表项,位于``和``标记之间。``有 `type` 和 `value` 两个常用属性。`type` 是数字样式,取值与``标记的 `type` 属性相同;`value` 指定新的数字序列起始值以获得非连续性数字序列。

(2) 无序列表标记

无序列表是一种在各列表项前面显示特殊项目符号的缩排列表,可以使用无序列表标记``和列表项标记``来创建,格式如下:

```
<ul 属性="值"...>
  <li>列表项 1</li>
  <li>列表项 2</li>
  ...
  <li>列表项 n</li>
</ul>
```

说明: 无序列表标记``常用的属性是 `type`,其取值为 `disc`、`circle` 和 `square`。它们分别表示用实心圆、空心圆和方块作为项目符号。

(3) 描述性列表标记

描述性列表标记`<dl>`和`<dd>`,本身并不具备作为列表显示的意义。只有当它们与``或``标签结构组合起来使用时,才能更好地表现出描述列表的作用。

比如,很多使用`<dl>`和`<dd>`来布局的网站的典型结构代码如下:

```
<div id="sidebar">
  <dl>
    <dt>栏目标题 1</dt>
    <dd>
      <ul>
        <li>新闻标题 1</li>
        ...
        <li>新闻标题 n</li>
      </ul>
    </dd>
    ...
  </div>
```

```

        <dt>栏目标题 n</dt>
        <dd>
            <ul>
                <li>新闻标题 1</li>
                ...
                <li>新闻标题 n</li>
            </ul>
        </dd>
    </dl>
</div>

```

这种简单的<dl>和<dd>组合更适合作为不同内容段的描述。

【例1.3】 创建一个有序列表，要求列表描述项的字体为黑体，斜体，颜色为红色，字号为4，列表项序列从 B 开始。

新建 1_3ex.html 文件，输入以下代码：

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
    <title>有序列表</title>
</head>
<body>
    <font face="黑体" color="red" size="4"><i>计算机课程</i></font>
    <ol type="A" start="2">
        <li>计算机导论</li>
        <li>操作系统</li>
        <li>计算机原理</li>
        <li>数据结构</li>
    </ol>
</body>
</html>

```

运行 1_3ex.html 文件，结果如图 1.6 所示。

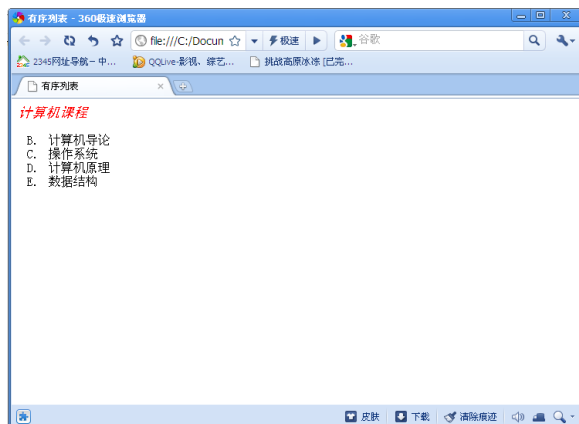


图 1.6 1_3ex.html 运行结果

1.2.3 XHTML 多媒体标记

1. 图像标记

利用图像标记可以向网页中插入图像，或者在网页中播放视频文件，格式如下：

```
<img 属性="值"... />
```

图像标记的属性如下。

- **src**: 图像文件的 URL 地址，图像可以是 jpeg、gif 或 png 文件。
- **alt**: 图像的简单说明，在浏览器不能显示图像或加载时间过长时显示。
- **height**: 所显示图像的高度（像素或百分比）。
- **width**: 所显示图像的宽度。
- **hspace**: 与左右相邻对象的间隔。
- **vspace**: 与上下相邻对象的间隔。
- **align**: 图像达不到显示区域大小时的对齐方式。当页面中有图像与文本混排时，可以使用此属性，取值为 top（顶部对齐）、middle（中央对齐）、bottom（底部对齐）、left（图像居左）、right（图像居右）。
- **border**: 图像边框像素数。
- **controls**: 指定该选项后，若有多媒体文件则显示一套视频控件。
- **dynsrc**: 指定要播放的多媒体文件。在标记中，dynsrc 属性要优先于 src 属性，如果指定的多媒体文件存在，则播放该文件，否则显示 src 指定的图像。
- **start**: 指定何时开始播放多媒体文件。
- **loop**: 指定多媒体文件播放次数。
- **loopdealy**: 指定多媒体文件播放之间的延迟（以 ms 为单位）。

例如：

```

```

说明：src="image/nj2014-1.jpg"是图像的相对路径，如果页面文件处于 Practice 文件夹，则说明该图像文件在 Practice 文件夹的 image 子文件夹下。

【例 1.4】用图像标记制作一个“2014 南京青奥会”主题的网页。

(1) 准备

新建 Practice 文件夹，在其下建立子目录 image；到南京青奥会官网去搜集两张图片，分别命名为 nj2014-1.jpg 和 nj2014-2.jpg；上网下载一个计时的动态 avi 文件，命名为 nj2014-clock.avi。两张图片和 avi 文件都存放在 Practice\image 路径下，如图 1.7 所示。

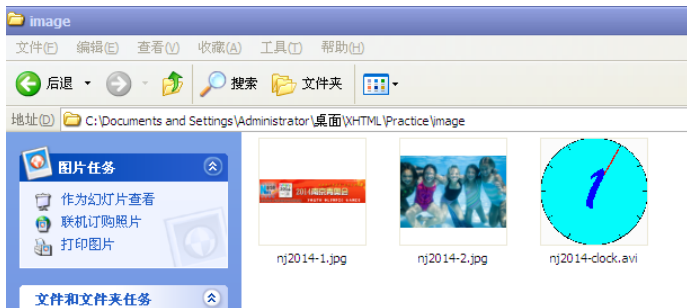


图 1.7 准备“2014 南京青奥会”主题网页的资源

(2) 编辑 XHTML 文档

输入下列内容，以 1_4nj2014.html 作为文件名保存：

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
    <title>欢迎南京青奥会</title>
</head>
<body>
    
    
    
</body>
</html>
```

(3) 运行

由于 360 安全浏览器会自动屏蔽掉 ActiveX 脚本，这样会使本例用到的多媒体文件 nj2014-clock.avi 无法正常播放，故建议大家使用普通的 IE 浏览器运行本网页，效果如图 1.8 所示。



图 1.8 “2014 南京青奥会”主题网页

2. 字幕标记

在 XHTML 语言中，可以在页面中插入字幕，水平或垂直滚动显示文本信息。字幕标记的格式如下：

```
<marquee 属性="值"...>滚动的文本信息</marquee>
```

说明：

<marquee>标记的主要属性如下。

- align: 指定字幕与周围主要属性的对齐方式, 取值为 top、middle、bottom。
- behavior: 指定文本动画的类型, 取值为 scroll (滚动)、slide (滑行)、alternate (交替)。
- bgcolor: 指定字幕的背景颜色。
- direction: 指定文本的移动方向, 取值为 down、left、right、up。
- height: 指定字幕的高度。
- hspace: 指定字幕的外部边缘与浏览器窗口之间的左右边距。
- vspace: 指定字幕的外部边缘与浏览器窗口之间的上下边距。
- loop: 指定字幕的滚动次数, 其值是整数, 默认为 infinite, 即重复显示。
- scrollamount: 指定字幕文本每次移动的距离。
- scrolleddelay: 指定前段字幕文本延迟多少毫秒后重新开始移动文本。

例如, 在【例 1.4】的文档 1_4nj2014.html 中添加如下几行代码。

```
<!DOCTYPE html
```

```
...>
```

```
<html>
```

```
<head>
```

```
...
```

```
</head>
```

```
<body>
```

```
...
```

```
    <marquee bgcolor="silver" direction="left" scrollamount="4" scrolledelay="100" width="700" height="30">
```

```
        <font face="黑体" size="5" color="red">
```

```
        <b>2014 年南京青奥会 让奥运走进青年, 让青年拥抱奥运! </b>
```

```
    </body>
```

```
</html>
```

再次运行, 显示效果如图 1.9 所示。



图 1.9 给网页添加滚动字幕

对比图 1.8 可以发现,网页下方多了一行滚动字幕:“2014 年南京青奥会 让奥运走进青年,让青年拥抱奥运!”。

3. 背景音乐标记

背景音乐标记只能放在文档头部分,也就是<head>与</head>标记之间,格式如下:

```
<bgsound 属性="值"... />
```

背景音乐标记的主要属性如下。

- **balance**: 指定将声音分成左声道和右声道,取值为-10 000~10 000,默认值为 0。
- **loop**: 指定声音播放的次数。设置为 0,表示播放一次;设置为大于 0 的整数,则播放指定的次数;设置为-1,表示反复播放。
- **src**: 指定播放的声音文件的 URL。
- **volume**: 指定音量高低,取值为-10 000~0,默认值为 0。

有兴趣的读者可以自己尝试给图 1.9 所示的网页加入声音特效。

1.2.4 XHTML 基本应用

XHTML 的基本应用包括表格的制作、表单的应用和超链接的应用等。

1. 表格的制作

一个表格由表头、行和单元格组成,常用于组织、显示信息或安排页面布局。一个表格通常由<table>标记开始,到</table>标记结束。表格的内容由<tr>、<th>和<td>标记定义。<tr>说明表的一个行,<th>说明表的列数和相应栏目的名称,<td>用来填充由<tr>和<th>标记组成的表格。

一个典型的表格格式如下:

```
<table 属性="值"...>
<caption>表格标题文字</caption>
<tr 属性="值"...>
    <th>第 1 个列表头</th> <th>第 2 个列表头</th>... <th>第 n 个列表头</th>
</tr>
<tr>
    <td 属性="值"...>第 1 行第 1 列数据</td>
    <td>第 1 行第 2 列数据</td>
    ...
    <td>第 1 行第 n 列数据</td>
</tr>
.....
<tr>
    <td>第 n 行第 1 列数据</td>
    <td>第 n 行第 2 列数据</td>
    ...
    <td>第 n 行第 n 列数据</td>
</tr>
</table>
```

(1) <table>标记的属性

用<table>标记创建表格时可以设置如下属性。

- **align**: 指定表格的对齐方式, 取值为 left (左对齐)、right (右对齐)、center (居中对齐), 默认值为 left。
- **background**: 指定表格背景图片的 URL 地址。
- **bgcolor**: 指定表格的背景颜色。
- **border**: 指定表格边框的宽度 (像素), 默认值为 0。
- **bordercolor**: 指定表格边框的颜色, border 不等于 0 时起作用。
- **bordercolordark**: 指定 3D 边框的阴影颜色。
- **bordercolorlight**: 指定 3D 边框的高亮显示颜色。
- **cellpadding**: 指定单元格内数据与单元格边框之间的间距。
- **cellspacing**: 指定单元格之间的间距。
- **width**: 指定表格的宽度。

(2) <tr>标记的属性

表格中的每一行都是由<tr>标记来定义的, 它有如下属性。

- **align**: 指定行中单元格的水平对齐方式。
- **background**: 指定行的背景图像文件的 URL 地址。
- **bgcolor**: 指定行的背景颜色。
- **bordercolor**: 指定行的边框颜色, 只有<table>标记的 border 属性不等于 0 时起作用。
- **bordercolordark**: 指定行的 3D 边框的阴影颜色。
- **bordercolorlight**: 指定行的 3D 边框的高亮显示颜色。
- **valign**: 指定行中单元格内容的垂直对齐方式, 取值为 top、middle、bottom、baseline (基线对齐)。

(3) <th>和<td>标记的属性

表格的单元格通过<td>标记来定义, 标题单元格可以使用<th>标记来定义, <th>和<td>标记的属性如下。

- **align**: 指定单元格的水平对齐方式。
- **bgcolor**: 指定单元格的背景颜色。
- **bordercolor**: 指定单元格的边框颜色, 只有<table>标记的 border 属性不等于 0 时起作用。
- **bordercolordark**: 指定单元格的 3D 边框的阴影颜色。
- **bordercolorlight**: 指定单元格的 3D 边框的高亮显示颜色。
- **colspan**: 指定合并单元格时一个单元格跨越的表格列数。
- **rowspan**: 指定合并单元格时一个单元格跨越的表格行数。
- **valign**: 指定单元格中文本的垂直对齐方式。
- **nowrap**: 若指定该属性, 则要避免 Web 浏览器将单元格的文本换行。

【例 1.5】 创建一个统计学生课程成绩的表格。

新建 1_5ex.html 文件, 输入以下代码:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>学生成绩显示</title>
```



```

</head>
<body>
<table align="center" border="1" bordercolor="red">
<caption><font size="5" color="blue">学生成绩表</font></caption>
  <tr bgcolor="#CCCCCC">
    <th width="80">专业</th>
    <th width="80">学号</th>
    <th width="80">姓名</th>
    <th width="90">计算机导论</th>
    <th width="90">数据结构</th>
  </tr>
  <tr>
    <td rowspan="3"><font color="blue">计算机</font></td>
    <td>081101</td>
    <td>王 &nbsp;&nbsp;林</td>
    <td align="center">80</td>
    <td align="center">78</td>
  </tr>
  <tr>
    <td>081102</td>
    <td>程 &nbsp;&nbsp;明</td>
    <td align="center">90</td>
    <td align="center">60</td>
  </tr>
  <tr>
    <td>081104</td>
    <td>韦 &nbsp;&nbsp;平</td>
    <td align="center">83</td>
    <td align="center">86</td>
  </tr>
  <tr>
    <td><font color="green">通信工程</font></td>
    <td>081201</td>
    <td>王 &nbsp;&nbsp;敏</td>
    <td align="center">89</td>
    <td align="center">100</td>
  </tr>
</table>
</body>
</html>

```

其运行结果如图 1.10 所示。

2. 表单的应用

表单用来从用户（站点访问者）处收集信息，然后将这些信息提交给服务器处理。表单中可以包含各种交互的控件，如文本框、列表框、复选框和单选按钮等。用户在表单中输入或选择数据后提交，该数据就会提交到相应的表单处理程序，以各种不同的方式进行处理。

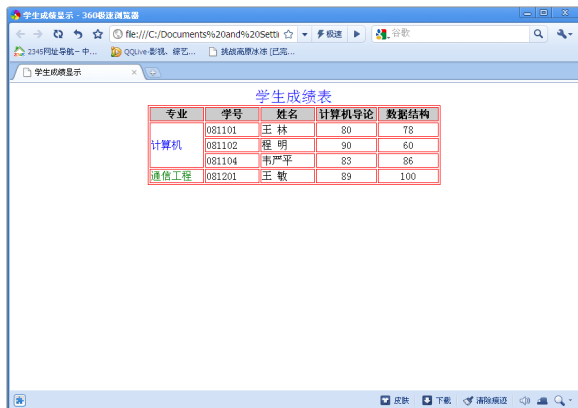


图 1.10 1_Sex.html 运行结果

表单定义格式如下：

```
<form 定义>
    [<input 定义 />]
    [<textarea 定义>]
    [<select 定义>]
    [<button 定义 />]
</form>
```

(1) 表单标记<form>

在 XHTML 语言中，表单内容用<form>标记来定义，格式如下：

```
<form 属性="值"...事件="代码">...</form>
```

➤ <form>标记的常用属性如下。

- **name**：指定表单的名称。命名表单后可以使用脚本语言来引用或控制该表单。
- **id**：指定表示该标记的唯一标识码。
- **method**：指定表单数据传输到服务器的方法，取值是 **post** 或 **get**。**post** 表示在 HTTP 请求中嵌入表单数据；**get** 表示将表单数据附加到该页请求的 URL 中。例如，某表单提交一个文本数据 **id** 值至 **page.htm** 页面。如果以 **post** 方法提交，新页面的 URL 为“**http://localhost/page.htm**”，而若以 **get** 方式提交相同表单，则新页面的 URL 为“**http://localhost/page.htm?id=...**”。
- **action**：指定接收表单数据的服务器端程序或动态网页的 URL 地址。提交表单之后，即运行该 URL 地址所指向的页面。
- **target**：指定目标窗口。**target** 属性的取值有 **_blank**、**_parent**、**_self** 和 **_top**，分别表示：在未命名的新窗口中打开目标文档；在显示当前文档的窗口的父窗口中打开目标文档；在提交表单所使用的窗口中打开目标文档；在当前窗口中打开目标文档。

➤ <form>标记的主要事件如下。

- **onsubmit**：提交表单时调用的事件处理程序。
- **onreset**：重置表单时调用的事件处理程序。

(2) 表单输入控件标记<input>

表单输入控件的格式如下：

```
<input 属性="值"... 事件="代码" />
```

为了让用户通过表单输入数据，在表单中可以使用<input>标记来创建各种输入型表单控

件。表单控件通过

① 单行文本框。在表单中添加单行文本框可以获取站点访问者提供的一行文本信息，格式如下：

```
<input type="text" 属性="值" ... 事件="代码" />
```

➤ 单行文本框的属性如下。

- name: 指定单行文本框的名称，通过它可以在脚本中引用该文本框控件。
- id: 指定表示该标记的唯一标识码，通过 id 值就可以获取该标记对象。
- value: 指定文本框的值。
- defaultvalue: 指定文本框的初始值。
- size: 指定文本框的宽度。
- maxlength: 指定允许在文本框内输入的最大字符数。
- form: 指定所属的表单名称（只读）。

例如，要设置如图 1.11 所示的文本框可以使用以下代码：

```
姓名: <input type="text" size="10" value="王小明" />
```

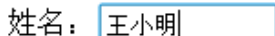


图 1.11 文本框

➤ 单行文本框的方法如下。

- Click(): 单击该文本框。
 - Focus(): 得到焦点。
 - Blur(): 失去焦点。
 - Select(): 选择文本框的内容。
- 单行文本框的事件如下。
- onclick: 单击该文本框执行的代码。
 - onblur: 失去焦点执行的代码。
 - onchange: 内容变化执行的代码。
 - onfocus: 得到焦点执行的代码。
 - onselect: 选择内容执行的代码。

② 密码框。密码框也是一个文本框，当访问者输入数据时，大部分浏览器会以星号显示密码，使别人无法看到输入内容，如图 1.12 所示，格式如下：

```
<input type="password" 属性="值"...事件="代码" />
```

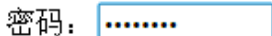


图 1.12 密码框

其中，属性、方法和事件与单行文本框基本相同，只是密码框没有 onclick 事件。

③ 隐藏域。在表单中添加隐藏域是为了使访问者看不到隐藏域的信息。每个隐藏域都有自己的名称和值。当提交表单时，隐藏域的名称和值就会与可见表单域的名称和值一起包含在

表单的结果中，格式如下：

```
<input type="hidden" 属性="值"... />
```

隐藏域的属性、方法和事件与单行文本框基本相同，只是没有 defaultvalue 属性。

④ 复选框。在表单中添加复选框是为了让站点访问者选择一个或多个选项，格式如下：

```
<input type="checkbox" 属性="值"...事件="代码" />选项文本
```

➤ 复选框的属性如下。

- name: 指定复选框的名称。
- id: 指定表示该标记的唯一标志码。
- value: 指定选中时提交的值。
- checked: 如果设置该属性，则第一次打开表单时该复选框处于选中状态。被选中时其值为 TRUE，否则为 FALSE。
- defaultchecked: 判断复选框是否定义了 checked 属性。已定义时其值为 TRUE，否则为 FALSE。

例如，要创建如图 1.13 所示的复选框，可以使用如下代码：

兴趣爱好：

```
<input type="checkbox" name="box" checked="checked" />旅游  
<input type="checkbox" name="box" checked="checked" />篮球  
<input type="checkbox" name="box" />上网
```

兴趣爱好： ☒ 旅游 ☒ 篮球 ☐ 上网

图 1.13 复选框

➤ 复选框的方法如下。

- Click(): 单击该复选框。
 - Focus(): 得到焦点。
 - Blur(): 失去焦点。
- 复选框的事件如下。
- onclick: 单击该复选框执行的代码。
 - onblur: 失去焦点执行的代码。
 - onfocus: 得到焦点执行的代码。

⑤ 单选按钮。在表单中添加单选按钮是为了让站点访问者从一组选项中选择其中一个选项。在一组单选按钮中，一次只能选择一个，格式如下：

```
<input type="radio" 属性="值" 事件="代码"... />选项文本
```

单选按钮的属性如下。

- name: 指定单选按钮的名称，若干名称相同的单选按钮构成一个控件组，在该组中只能选择一个选项。
- value: 指定提交时的值。
- checked: 如果设置了该属性，当第一次打开表单时该单选按钮处于选中状态。

单选按钮的方法和事件与复选框相同。

当提交表单时，该单选按钮组名称和所选取的单选按钮指定值都会包含在表单结果中。

例如，要创建如图 1.14 所示的单选按钮，可以使用如下代码：

```
<input type="radio" name="rad" value="1" checked="checked" />男
<input type="radio" name="rad" value="0" />女
```

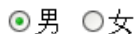


图 1.14 单选按钮

⑥ 按钮。使用<input>标记可以在表单中添加三种类型的按钮：“提交”按钮、“重置”按钮和“自定义”按钮，格式如下：

```
<input type="按钮类型" 属性="值" onclick="代码" />
```

其中，事件 onclick 的值是单击按钮后执行的脚本代码。

根据 type 值的不同，按钮的类型也不一样。

- type="submit": 创建一个“提交”按钮。单击该按钮，表单数据（包括提交按钮的名称和值）会以 ASCII 文本形式传送到由表单的 action 属性指定的表单处理程序中。一般来说，一个表单必须有一个“提交”按钮。
- type="reset": 创建一个“重置”按钮。单击该按钮，将删除所有已经输入表单中的文本并清除所有选择。如果表单中有默认文本或选项，将会恢复这些值。
- type="button": 创建一个“自定义”按钮。在表单中添加自定义按钮时，必须为该按钮编写脚本以使按钮执行某种指定的操作。

按钮的其他属性还有 name（按钮的名称）和 value（显示在按钮上的标题文本），例如：

```
<input type="submit" name="bt1" value="提交" />
<input type="reset" name="bt2" value="重置" />
<input type="button" name="bt3" value="自定义" />
```

⑦ 文件域。文件域由一个文本框和一个“浏览”按钮组成，用户可以在文本框中直接输入文件的路径和文件名，或单击“浏览”按钮从磁盘上查找、选择所需文件，格式如下：

```
<input type="file" 属性="值"...>
```

文件域的属性有 name（文件域的名称）、value（初始文件名）和 size（文件名输入框的宽度）。例如，要创建如图 1.15 所示的文件域（普通 IE 浏览器的显示效果），可以使用如下代码：

```
<input type="file" name="fl" size="20" />
```



图 1.15 文件域

（3）其他表单控件

① 滚动文本框。在表单中添加滚动文本框是为了使访问者可以输入多行文本，格式如下：

```
<textarea 属性="值"...事件="代码"...>初始值</textarea>
```

说明：<textarea>标记的属性有 name（滚动文本框控件的名称）、rows（控件的高度，以行为单位）、cols（控件的宽度，以字符为单位）和 readonly（滚动文本框中的内容是否能被修改）。

滚动文本框的其他属性、方法和事件与单行文本框基本相同。

例如，要创建如图 1.16 所示的滚动文本框（普通 IE 浏览器的显示效果），可以使用如下代码：

```
<meta http-equiv="content-type" content="text/html; charset=gb2312">
```

```
<textarea name="ta" rows="8" cols="20" readonly="readonly">这是本文本框的初始内容，是只读的，用户无法修改</textarea>
```

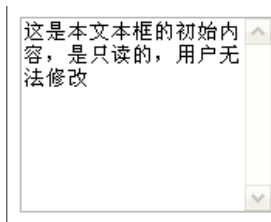


图 1.16 滚动文本框

② 选项选单。表单中选项选单（下拉菜单）的作用是使访问者从列表或选单中选择选项，格式如下：

```
<select name="值" size="值" [multiple="multiple"]>
  <option [selected="selected"] value="值">选项 1</option>
  <option [selected="selected"] value="值">选项 2</option>
  ...
</select>
```

其中：

- **name**：指定选项选单控件的名称。
- **size**：指定在列表中一次可看到的选项数目。
- **multiple**：指定允许做多项选择。
- **selected**：指定该选项的初始状态为选中。

例如，要创建如图 1.17 所示的选项选单，可以使用如下代码：

```
学历：<select name="se" size="1" >
  <option>研究生</option>
  <option selected="selected">大学</option>
  <option>高中</option>
  <option>初中</option>
  <option>小学</option>
</select>
```

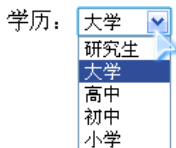


图 1.17 选项选单

③ 对表单控件进行分组。可以使用<fieldset>标记对表单控件进行分组，将表单划分为更小、更易于管理的部分，格式如下：

```
<fieldset>
  <legend>控件组标题</legend>
  组内表单控件
</fieldset>
```

【例 1.6】制作一个学生个人资料的表单，包括姓名、学号、性别、出生日期、所学专业、选修课程、备注和兴趣等信息项。要求综合运用前面所讲的各类表单控件，使得页面简洁美观。

创建文件 1_6stu.html，输入以下代码：

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
    <title>学生个人信息</title>
</head>
<body>
<form name="form1" method="post" action="xsServlet">
<fieldset style="width:450px" align="center">
<legend><b>学生个人信息</b></legend>
    <fieldset style="width:400px" align="left">
        <legend><b>基本资料</b></legend>
        <table width="400" border="0" align="center" bgcolor="#CCFFCC">
            <tr>
                <td width="120">学号： </td>
                <td><input name="XH" type="text" value="081101"></td>
            </tr>
            <tr>
                <td>姓名： </td>
                <td><input name="XM" type="text" value="王林"></td>
            </tr>
            <tr>
                <td>性别： </td>
                <td><input name="SEX" type="radio" value="男" checked="checked">男
                    <input name="SEX" type="radio" value="女">女</td>
            </tr>
            <tr>
                <td>出生日期： </td>
                <td><input name="Birthday" type="text" value="1989-05-06"></td>
            </tr>
        </table>
    </fieldset>
    <fieldset style="width:400px" align="left">
        <legend><b>详细资料</b></legend>
        <table width="400" border="0" align="center" bgcolor="#CCFFFF">
            <tr>
                <td>专业： </td>
                <td><select name="ZY">
                    <option>计算机</option>
                    <option>软件工程</option>
                    <option>信息管理</option>
                    <option>通信工程</option>
```

```

        <option>信息网络</option>
    </select></td>
</tr>
<tr>
    <td>选修课程: </td>
    <td><select name="KC" size="3" multiple="multiple">
        <option selected>计算机导论</option>
        <option selected>数据结构</option>
        <option>数据库原理</option>
        <option>操作系统</option>
        <option>计算机网络</option>
    </select></td>
</tr>
<tr>
    <td>备注: </td>
    <td><textarea name="BZ">团员</textarea></td>
</tr>
<tr>
    <td>兴趣: </td>
    <td><input name="XQ" type="checkbox" value="听音乐" checked="checked" >听音乐
        <input name="XQ" type="checkbox" value="看小说">看小说
        <input name="XQ" type="checkbox" value="上网" checked="checked">上网</td>
</tr>
</table>
</fieldset>
<input type="submit" name="BUTTON1" value="提交" align="center">
<input type="reset" name="BUTTON2" value="重置" align="center">
</fieldset>
</form>
</body>
</html>

```

运行结果如图 1.18 所示。



图 1.18 1_6stu.html 运行结果

3. 超链接的应用

在网页中,超链接通常以文本或图像形式呈现。当鼠标指针指向网页中的超链接时,会变成手的形状。单击超链接,浏览器会按照超链接所指示的目标载入另一个网页,或者跳转到同一网页的其他位置,格式如下:

```
<a 属性="值"...>超链接内容</a>
```

按照目标地址的不同,超链接分为文件链接、锚点链接和邮件链接。

(1) 文件链接

文件链接的目标地址是网页文件,目标网页文件可以位于当前服务器或其他服务器上。超链接使用<a>标记来创建,其常用的属性如下。

- **href**: 指定目标地址的 URL,这是必选项。
- **target**: 指定窗口或框架的名称。该属性指定将目标文档在指定的窗口或框架中打开。如果省略该属性,则在当前窗口中打开。**target** 属性的取值可以是窗口或框架的名称,也可以是如下保留字。

_blank: 未命名的新浏览器窗口。

_parent: 父框架或窗口。

_self: 所在的同一窗口或框架。

_top: 整个浏览器窗口中,并删除所有框架。

- **title**: 指定超链接时所显示的标题文字,例如:

```
<a href="http://www.qq.com">腾讯</a>
```

```
<a href="1_6stu.html">链接到本文件夹中的 1_6stu.html 文件</a>
```

```
<a href=" ../index.html">链接到上一级文件夹中的 index.html 文件</a>
```

```
<a href="image/tp.jpeg">链接到图片</a>
```

```
<a href="http://www.163.com" title="图片链接"></a>
```

(2) 锚点链接

锚点链接的目标地址是网页中的一个位置。创建锚点链接时,要在页面的某一处设置一个位置标记(锚点),并给该位置指定一个名称,以便在同一页面或其他页面中引用。

要创建锚点链接,首先要在页面中用<a>标记为要跳转的位置命名,例如,在 1_6stu.html 页面中进行如下设置:

```
<a id="xlxq"></a>
```

说明: <a>和标记之间不要放置任何文字。

创建锚点后如果在同一页面中要跳转到名为“xlxq”的锚点处,可以使用如下代码:

```
<a href="#xlxq">去本页面的锚点处</a>
```

如果要从其他页面跳转到该页面的锚点处,可以使用如下代码:

```
<a href="1_6stu.html #xlxq">去该页面的锚点处</a>
```

(3) 邮件链接

通过邮件链接可以启动电子邮件客户端程序,并由访问者向指定地址发送邮件。创建邮件链接也使用<a>标记,该标记的 href 属性由三部分组成:电子邮件协议名称 **mailto**,电子邮件地址,可选的邮件主题(其形式为“**subject**=主题”)。前两部分之间用冒号分隔,后两部分之间用问号分隔,例如:

```
<a href="mailto:163@163.com?subject=XHTML 教程">当前教程答复</a>
```

当访问者在浏览器窗口中单击邮件链接时,会自动启动电子邮件客户端程序,并将指定的

可以看到, 图 1.19 的页面上多了三个超链接: “[南京大学](#)”和“[东南大学](#)”两个文字链接、[南京师范大学](#)徽标图片链接。可以试着单击这些链接, 看看它们都去往哪里。

1.2.5 框架网页设计

框架可以将文档划分为若干窗格, 在每个窗格中显示一个网页, 从而得到在同一个浏览器窗口中显示不同网页的效果。框架网页是通过一个框架集<frameset>和多个框架<frame>标记来定义的。在框架网页中将<frameset>标记放在<head>标记之后取代<body>的位置, 还可以使用<noframes>标记指出框架不能被浏览器显示时的替换内容。

框架网页的基本结构如下:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html>
<head>
    <title>框架网页的基本结构</title>
</head>
<frameset 属性="值"...>
    <frame 属性="值"... />
    <frame 属性="值"... />
    ...
</frameset>
</html>
```

1. 框架集

框架集包括如何组织各个框架的信息, 可以用<frameset>标记定义。框架是按照行、列组织的, 可以用<frameset>标记的下列属性对框架结构进行设置。

- cols: 在创建纵向分隔框架时指定各个框架的列宽, 取值有三种形式, 即像素、百分比和相对尺寸。例如:

cols="*, *, *"表示将窗口划分为三个等宽的框架。

cols="30%, 200, *"表示将浏览器窗口划分为三列框架, 其中第 1 列占窗口宽度的 30%, 第 2 列为 200 像素, 第三列为窗口的剩余部分。

cols="*, 3 *, 2 *"表示左边的框架占窗口的 1/6, 中间的占 1/2, 右边的占 1/3。

- rows: 指定横向分隔框架时各个框架的行高, 取值与 cols 属性类似。但 rows 属性不能与 cols 属性同时使用, 若要创建既有纵向分隔又有横向分隔的框架, 应使用嵌套框架。
- frameborder: 指定框架周围是否显示 3D 边框。若取值为 1 (默认值) 则显示, 为 0 则显示平面边框。
- framespacing: 指定框架之间的间隔 (以像素为单位, 默认为 0)。

要创建一个嵌套框架集, 可以使用如下代码:

```
<html>
<head>
    <title>嵌套框架</title>
```

```

</head>
<frameset rows="20%,400,*">
    <frame />
    <frameset cols="300,*" />
        <frame />
        <frame />
    </frameset>
</frameset>
</html>

```

2. 框架

框架使用<frame>标记来创建，主要属性如下。

- name: 指定框架的名称。
- frameborder: 指定框架周围是否显示 3D 边框。
- marginheight: 指定框架的高度（以像素为单位）。
- marginwidth: 指定框架的宽度（以像素为单位）。
- noresize: 指定不能调整框架的大小。
- scrolling: 指定框架是否可以滚动，取值为 yes、no 和 auto。
- src: 指定在框架中显示的网页文件。

【例 1.8】设计一个框架网页，并在各个框架中各显示一个网页。

(1) 1_8frame.html（主网页）

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html>
<head>
    <title>框架中显示网页</title>
</head>
<frameset rows="80,*">
    <frame src="1_8top.html" name="frmtop" />
    <frameset cols="25%,*">
        <frame src="1_8left.html" name="frmleft" />
        <frame src="1_8content.html" name="frmmain" />
    </frameset>
</frameset>
</html>

```

(2) 1_8top.html（上部网页）

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<body bgcolor="#8888FF">
<marquee behavior="alternate" direction="right">
    <font size="5" color="blue">欢迎登录学生成绩管理系统</font>

```

```

</marquee>
</body>
</html>

```

(3) 1_8content.html (下部右边网页)

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
    <title>content 网页</title>
</head>
<body>
    <h2 align="center">这里是 content 网页。</h2>
</body>
</html>

```

(4) 1_8left.html (下部左边网页)

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
    <title>left 网页</title>
</head>
<body>
    <a href="1_5ex.html" target="frmmain">学生成绩表</a><br><br>
    <a href="1_6stu.html" target="frmmain">学生信息显示</a><br><br>
    <a href="1_8content.html" target="frmmain">返回主页</a><br>
</body>
</html>

```

完成后运行 1_8frame.html 文件，单击页面下部左边网页的“学生信息显示”超链接，运行效果如图 1.20 所示。



图 1.20 框架网页

1.3 CSS 初步

层叠样式表（Cascading Style Sheets, CSS）是 W3C 协会为弥补 XHTML 在显示方面的不足而制定的一套扩展样式标准。CSS 标准重新定义了 XHTML 中的文字显示样式，并增加了一些新概念，提供了更为丰富的显示样式。同时，CSS 还可进行集中样式管理，允许将样式定义单独存储于样式文件中，这样可以使显示内容和显示样式定义分离，使多个 XHTML 文件共享样式定义。

1.3.1 CSS 定义及引用

样式表的作用是告诉浏览器如何呈现文档，样式定义是 CSS 的基础。通常，CSS 可以通过三种方式对页面中的元素进行样式定义：内嵌样式、内部样式和外联样式。

下面只简单介绍一下这三种样式定义在网页中的应用。

1. 内嵌样式

在标记中直接使用 style 属性可以对该标记括起的内容应用样式来显示，例如：

```
<p style="font-family: '宋体';color:green;background-color:yellow;font-size:9px"></p>
```

使用 style 属性定义时，内容与值之间用冒号“:”分隔。用户可以定义多项内容，内容之间以分号“;”分隔。由于这种方式在 XHTML 标记内部引用样式，所以称为内嵌样式或内联样式。

注意：若要在 XHTML 文件中使用内嵌样式，必须在该文件的头部对整个文档进行单独的样式语言声明，例如：

```
<meta http-equiv="Content-type" content="text/css; charset=gb2312" />
```

由于内嵌样式将样式和要展示的内容混在一起，违背了使用样式表的初衷，所以建议尽量不要使用这种方式。

2. 内部样式

所谓的内部样式，就是利用 style 标签来包含本页所需样式定义的代码。它虽然也是将表现样式的代码和组织内容的代码放在同一个页面中，但是由于其单独将表现样式的 CSS 代码放在 style 标签之内，故它与内嵌样式有着本质上的区别。

定义内部样式表的格式如下：

```
.类选择符{规则表}
```

其中，“类选择符”是引用的样式的类标记，“规则表”是由一个或多个样式属性组成的样式规则，各样式属性间用分号隔开，每个样式属性的定义格式为“样式名:值”。例如：

```
.style1{font-family:"黑体"; color:green; font-size:15px;}
```

其中，“font-family”表示字体，“color”表示字体颜色，“font-size”表示字体大小。样式表定义时使用<style>标记括起，放在<head>标记范围内，<style>标记内定义的前后可以加上注释符“<!--”、“-->”，它的作用是使不支持 CSS 的浏览器忽略样式表定义。<style>标记的 type 属性指明样式的类别，默认值为“text/css”，例如：

```
<head>
  <style type="text/css">
    <!--
```

```

        .style1 {font-size: 20px; font-family: "黑体";}
    -->
    </style>
</head>

```

内部样式表主要使用标记的 `class` 属性来引用, 只要将标记的 `class` 属性值设置为样式表中定义的类选择符即可, 例如:

```

<div class="style1">内部样式表的引用</div>
<input type="text" name="text" class="style1" />

```

利用类选择符和标记的 `class` 属性, 可以使相同的标记使用不同的样式, 或使不同的标记使用相同的样式。

【例 1.9】 内嵌样式与内部样式示例。

输入下列内容, 以 `1_9css1.html` 作为文件名保存:

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
    <title>CSS 示例</title>
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
    <style type="text/css">
    <!--
        .heiti {font-size: 20px; font-family: "黑体"; color:red;}
    -->
    </style>
</head>
<body>
    <div>
    <p style="font-family: '宋体';color:green;background-color:yellow;font-size:9px">内嵌样式</p>
    </div>
    <div class="heiti">内部样式</div>
    <input type="text" name="text" class="heiti" />
</body>
</html>

```

运行文件, 将显示如图 1.21 所示的页面。

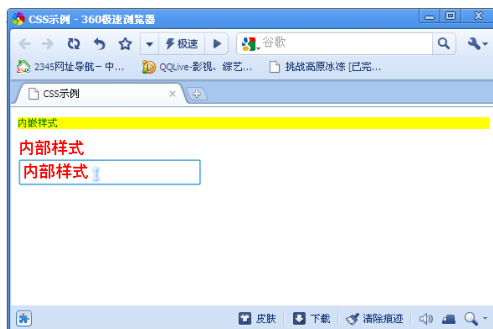


图 1.21 内嵌样式与内部样式

3. 外联样式

无论是内嵌样式还是内部样式，都只能由当前的 XHTML 文档引用，这样一来，只有当前页面中的元素可以重用 CSS 代码，而其他页面则不能，这对于制作大型网站是极为不利的！因为大网站往往囊括了数量庞大的页面，且众多页面的显示风格是高度一致的，大型网站的这些特点对 CSS 代码重用提出了更高的要求，需要依靠外联样式。

外联样式表就是把样式存放在单独的 CSS 文件中。在 XHTML 中的<head>中采用<link>标记把 CSS 文件关联起来，例如：

```
<head>
<meta ... />
<link href="mystyle.css" type="text/css" rel="stylesheet" rev="stylesheet" />
</head>
```

其中，mystyle.css 是定义的样式表文件，内容形如：

```
div{
    width:300px;           /*定义 div 元素的宽度为 300 像素*/
    height:200px;          /*定义 div 元素的高度为 200 像素*/
    padding:6px;
    border:#006600 2px solid;
    font-size:16px;
    color:#889900;
}
#styl1{
    ...
}
...
```

这样，被关联的 XHTML 中的 div 均采用该样式，也可以采用 class 属性引用其他样式。

引用样式文件的 XHTML 文档在头部用<link>标记链接 CSS 样式文件，<link>标记的属性主要有 rel、href、type 和 media。rel 属性用于定义链接的文件和 XHTML 文档之间的关系，通常取值为 stylesheet；href 属性指出 CSS 样式文件的位置和文件名；type 属性指出样式的类别（通常取值为 text/css）；media 属性用于指定接受样式表的介质，默认值为 screen（显示器），还可以是 print（打印机）、projection（投影机）等。

【例 1.10】XHTML 文档链接 CSS 文件。

（1）定义独立的 CSS 样式文件

外联样式定义的内容一般放在一个独立的 CSS 样式文件中，本例取文件名为 1_10style1.css，内容如下：

```
p {font-family: "宋体"; color: green; background-color: yellow; font-size: 12pt; }
h1,h2 {font-family: "隶书", "宋体"; color:#ff8800}
.heti {font-family: "黑体"; font-size: 20pt; color: #000000; }
#id1 {color: blue; }
```

注意：文件不包含<style>标记，因为<style>是 html 标记而非 css 样式。

（2）编辑 XHTML 文档，链接 CSS 文件

输入下列内容，以 1_10css2.html 作为文件名保存：

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```



```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>链接外部 css 文件</title>
  <meta http-equiv="content-type" content="text/html; charset=gb2312">
  <link rel="stylesheet" type="text/css" href="1_10style1.css" media="screen">
</head>
<body topmargin=4>
  <h1>内容 h1 样式显示</h1>
  <h2>内容 h2 样式显示</h2>
  <h3 id="id1">内容 id1 样式显示</h3>
  <h4>h4 内容默认样式显示</h4>
  <p>内容 p 样式显示</p>
  <p class="heti">内容 heti 样式显示</p>
</body>
</html>
```

用浏览器打开文档，将显示如图 1.22 所示的页面。

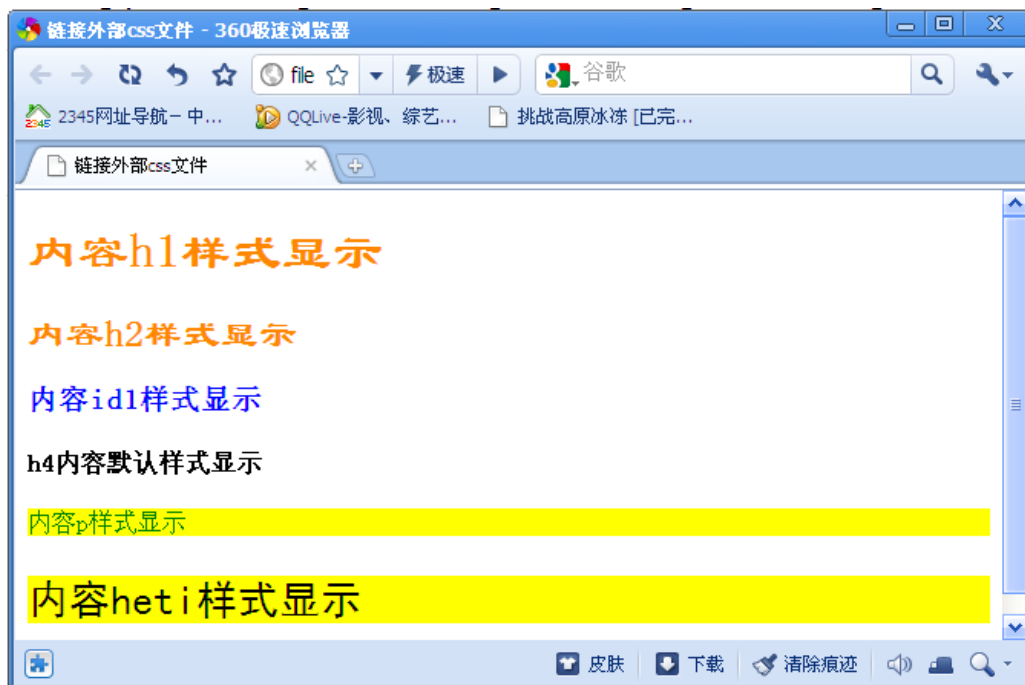


图 1.22 链接 CSS 文件

细心的读者会注意到本例文件 1_10style1.css 中 p{}、h1,h2{}、#id1{} 的用法与前述“.类选择符{规则表}”的格式略有差异，下面的 1.3.2 节将对此详加说明。

1.3.2 CSS 选择符

定义样式表的符号就是 CSS 选择符。选择符可分为以下几种情况。

1. 标记符

标记符{规则表}

标记符可以是一个或多个，各个标记之间以逗号分开。

例如，【例 1.10】的 CSS 文件里有如下代码：

```
p {font-family: "宋体"; color: green; background-color: yellow; font-size: 12pt; }  
h1,h2 {font-family: "隶书", "宋体"; color:#ff8800}
```

2. 类选择符和 class 属性

利用类选择符和标记的 class 属性可以使相同的标记使用不同的样式，也可以使不同的标记使用同样的样式，因为只要使标记的 class 属性值为样式表中定义的类选择符即可。

类选择符在样式表中定义具有样式值的类，有两种定义格式：

- ① 标记名.类名{规则 1; 规则 2; ... }
- ② .类名{规则 1; 规则 2; ... }

前者是为**特定的标记**定义的类，该标记的 class 属性设置为该类名，**只有使用该标记的内容才会采用这个样式**；后者为一般定义的类，只要某标记引用了该类名就可采用这个样式。请看下面的例子。

【例 1.11】两种 CSS 类选择符示例。

输入下列内容，以 1_11css3.html 作为文件名保存：

```
<!DOCTYPE html  
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head>  
  <title>两种 css 类选择符</title>  
  <meta http-equiv="content-type" content="text/html; charset=gb2312">  
  <style type="text/css">  
    <!--  
      p.back{font-family:"隶书", "宋体"; color:#ff8800}  
      .heti {font-family:"黑体"; font-size: 20pt; color:#000000;}  
    -->  
  </style>  
</head>  
<body topmargin=4>  
  <p class="back">标签 p 的内容可以 p.back 样式显示</p>  
  <div class="back">标签 div 的内容不可以 p.back 样式显示</div>  
  <p class="heti">标签 p 的内容可以.heti 样式显示</p>  
  <div class="heti">标签 div 的内容也可以.heti 样式显示</div>  
</body>  
</html>
```

运行文件，将显示如图 1.23 所示的页面。

3. id 选择符和 id 属性

id 选择符用于定义一个元素独有的样式，它与类选择符的区别在于：id 选择符在一个 XHTML 文件中只能引用一次，而类选择符可多次引用。

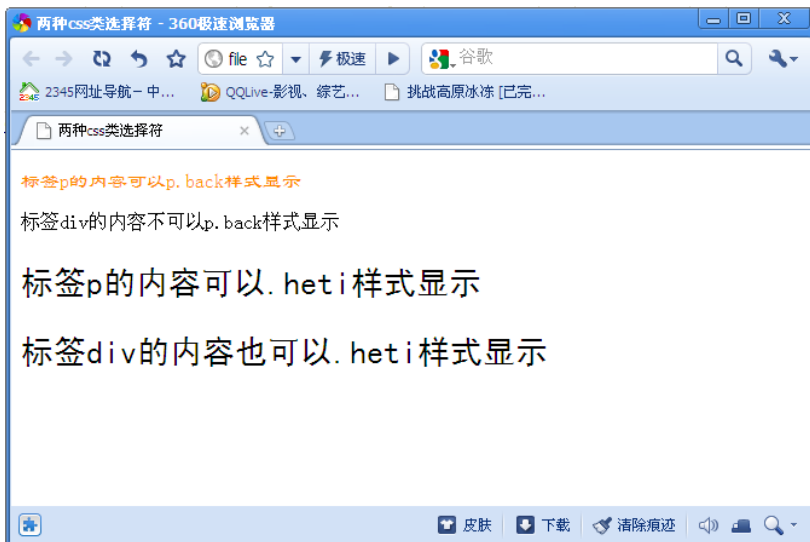


图 1.23 两种 CSS 类选择符的适用对象

id 选择符的定义格式如下：

```
#id 名{规则 1; 规则 2;... }
```

这个选择符在【例 1.10】的 CSS 文件中曾经出现过：

```
#id1 {color: blue; }
```

其引用方法如下：

```
<h3 id="id1">内容 id1 样式显示</h3>
```

1.3.3 CSS 属性

CSS 属性包括字体属性、颜色和背景属性、文本属性、列表属性、方框属性、分类属性和定位属性等。

1. 字体属性

字体属性如表 1.3 所示。

表 1.3 字体属性说明

属 性 名	取 值	说 明
font-family	“宋体” “隶书” ...	字体
font-size	12pt ... 8px ...	字号
font-style	italic bold ...	字体风格
font-weight	100 200 ...	字加粗
font-variant		字体变化
font		字体综合设置

2. 颜色和背景属性

颜色和背景属性如表 1.4 所示。

表 1.4 颜色和背景属性说明

属 性 名	取 值	说 明
color	颜色表示	指定页面元素的前景色
background-color	颜色表示 transparent	指定页面元素的背景色
background-image	URL none	指定页面元素的背景图像
background-repeat	repeat repeat-x repeat-y no-repeat	指定一个被指定的背景图像被重复的方式（默认值为 repeat）
background-attachment	scroll fixed	指定背景图像是否跟随页面内容滚动（默认值为 scroll）
background-position	数值表示法 关键词表示法	指定背景图像的位置
background	背景颜色 背景图像 背景重复 背景位置	背景属性综合设定

3. 文本属性

文本属性如表 1.5 所示。

表 1.5 文本属性说明

属 性 名	取 值	说 明
letter-spacing	长度值 normal	设定字符之间的间距
text-decoration	none underline overline line-through blink	设定文本的修饰效果（默认值为 none）， line-through 是删除线，blink 是闪烁效果
text-align	left right center justify（将文字均分展开）	设置文本横向排列对齐方式

续表

属 性 名	取 值	说 明
vertical-align	baseline super Sub top middle bottom text-top text-bottom 百分比	设定元素纵向排列对齐方式
text-indent	长度值 百分比	设定块级元素第一行的缩进量
line-height	normal 长度值 数字 百分比	设定相邻两行的间距

4. 列表属性

列表属性如表 1.6 所示。

表 1.6 列表属性说明

属 性 名	取 值	说 明
list-style-type	无序列表值： disc circle square 有序列表值： decimal lower-roman upper-roman lower-alpha upper-alpha 公用值：none	表项的项目符号。 disc：实心圆点 circle：空心圆 square：实心方形 decimal：阿拉伯数字 lower-roman：小写罗马数字 upper-roman：大写罗马数字 lower-alpha：小写英文字母 upper-alpha：大写英文字母 none：不设定
list-style-image	url（URL）	使用图像作为项目符号
list-style-position	outside、inside	设置项目符号是否在文字里面，与文字对齐
list-style	项目符号、位置	综合设置项目属性

5. 方框属性

方框属性如表 1.7 所示。

表 1.7 方框属性说明

属 性 名	说 明
margin-top	设定 HTML 文件内容与块元素的上边界距离。值为百分比时依照其上级元素的设置值（默认值为 0）
margin-right	设定 HTML 文件内容与块元素的右边界距离
margin-bottom	设定 HTML 文件内容与块元素的下边界距离
margin-left	设定 HTML 文件内容与块元素的左边界距离
margin	设定 HTML 文件内容与块元素的上、右、下、左边界距离。如果只给出 1 个值，则被应用于 4 个边界，如果只给出 2 个或 3 个值，则未显式给出值的边用其对边的设定值
padding-top	设定 HTML 文件内容与上边框之间的距离
padding-right	设定 HTML 文件内容与右边框之间的距离
padding-bottom	设定 HTML 文件内容与下边框之间的距离
padding-left	设定 HTML 文件内容与左边框之间的距离
padding	设定 HTML 文件内容与上、右、下、左边框的距离。设定值的个数与边框的对应关系同 margin 属性
border-top-width	设置元素上边框的宽度
border-right-width	设置元素右边框的宽度
border-bottom-width	设置元素下边框的宽度
border-left-width	设置元素左边框的宽度
border-width	设置元素上、右、下、左边框的宽度。设定值的个数与边框的对应关系同 margin 属性
border-top-color	设置元素上边框的颜色
border-right-color	设置元素右边框的颜色
border-bottom-color	设置元素下边框的颜色
border-left-color	设置元素左边框的颜色
border-color	设置元素上、右、下、左边框的颜色。设定值的个数与边框的对应关系同 margin 属性
border-style	设定元素边框的样式。设定值的个数与边框的对应关系同 margin 属性（默认值为 none）
border-top	设定元素上边框的宽度、样式和颜色
border-right	设定元素右边框的宽度、样式和颜色
border-bottom	设定元素下边框的宽度、样式和颜色
border-left	设定元素左边框的宽度、样式和颜色
width	设置元素的宽度
height	设置元素的高度
float	设置文字围绕于元素周围。left: 元素靠左，文字围绕在元素右边；right: 元素靠右，文字围绕在元素左边；None: 以默认位置显示
clear	清除元素浮动。none: 不取消浮动；left: 文字左侧不能有浮动元素；right: 文字右侧不能有浮动元素；both: 文字两侧都不能有浮动元素

6. 定位属性

定位属性如表 1.8 所示。

表 1.8 定位属性说明

属 性 名	说 明
top	设置元素与窗口上端的距离
left	设置元素与窗口左端的距离
position	设置元素位置的模式
z-index	z-index 将页面中的元素分成多个“层”，形成多个层“堆叠”，从而营造出三维空间效果

1.4 动 态 网 页

1.4.1 何谓“动态”网页

网络技术日新月异，细心的网友会发现许多网页文件扩展名不再只是“.htm”、“.html”，还有“.php”、“.asp”、“.jsp”等，这些都是采用动态网页技术制作出来的。

动态网页与传统静态网页在表观效果上有着显著的不同，它的最大特点是交互性和智能性很强。采用动态网页技术的网站可以实现更多的功能，如用户注册、用户登录、在线调查、用户管理、订单管理等。

如图 1.24 所示为国内最大购物网站“淘宝网”的登录页。



图 1.24 淘宝网登录页

在今天的 Web 2.0 时代，几乎所有的网站都是“动态”的了，它们以各大互联网公司背后强大的服务器集群和数据库系统为支撑，每天都“接待”数以亿计的用户，为人们的网上活动提供便利。

归纳起来，动态网页具有如下两个根本特征。

① 动态网页一般以数据库技术为基础，可以大大降低网站维护的工作量。

② 动态网页实际上并不是独立存在于服务器上的网页文件，只有当用户请求时，服务器才返回一个完整的网页。

这里说的动态网页，与网页上的各种动画、滚动字幕等视觉上的“动态效果”没有直接关系，如本章【例 1.4】“2014 南京青奥会”的主题网页，虽然我们用字幕标记在页面下方添加了一行游动的字幕，具有了动态效果，但它仍然属于静态网页。

动态网页可以是纯文字内容的，也可以是包含各种动画内容的，这些只是网页具体内容的表现形式，无论网页是否具有动态效果，只有采用动态网站技术生成的网页才可称之为动态网页。

1.4.2 动态网站架构原理

动态网站普遍是采用分布式计算模式架构起来的，使用这种模式构建网络应用时，通常需要开发大量的程序，这些程序部署在不同的计算机上，在应用中承担着不同的职责。例如，有的程序展示用户界面，有的程序进行逻辑计算，有的则进行后台数据处理。

我们通常把这样的应用系统分为多层结构，其中最典型的是 B-S-D（Browse/Server/Database Server，浏览器/服务器/数据库）3 层结构（见图 1.25）。

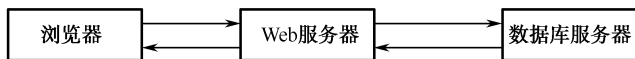


图 1.25 动态网站系统的 3 层结构

第 1 层为浏览器。浏览器是用户输入数据和显示结果的交互界面，用户在浏览器表单中输入数据，然后将表单中的数据提交并发送到 Web 服务器。

第 2 层为 Web 服务器。Web 服务器应用程序接收并处理用户的数据，并从数据库中查询用户数据，或者把用户数据录入数据库。最后 Web 服务器把返回的结果插入 XHTML 页面，传送回客户端，并在浏览器中显示出来。

第 3 层为数据库服务器。数据库服务器储存网站的海量用户信息，并提供数据处理和事务处理功能。

在某些规模不是很大的网站应用中，也有把 Web 服务器和数据库服务器（软件）安装在同一台计算机上的，在这种情况下，第 2 层和第 3 层就合并为同一层，统称为“服务器端”。但无论是典型的 3 层结构或简化的 2 层结构，在广大的网民用户看来，“服务器端”都是“透明”的，都是 B/S 结构。

1.4.3 Web 开发工具

早期的动态网页主要采用 CGI 技术，CGI 即 Common Gateway Interface（公用网关接口）。可以使用不同的语言编写适合的 CGI 程序，如 Visual Basic、Delphi 或 C/C++ 等。虽然 CGI 技术已经发展成熟而且功能强大，但由于编程困难、效率低下、修改复杂，所以已经逐渐被新技术取代。

现在的 Web 开发工具不断呈现，各自的功能特点也不尽相同，这些工具主要可以方便用

户设计网页和网页中用到的各种元素。目前，常用的客户端网页设计工具包括 FrontPage、Dreamweaver、Fireworks 和 Flash 等，而服务器端常用的动态网页交互技术有 ASP(Active Server Pages)、ASP.NET、JSP (Java Server Pages)、PHP (Hypertext Preprocessor) 等，它们都提供在 XHTML 代码中混合某种程序代码、由语言引擎解释执行程序代码的能力。本书将要介绍的是目前最为主流的动态网页开发技术——JSP。

1.5 上机练习

1. 设计网页，文件名为 my_college.html，如图 1.26 所示（与图 1.3 相同）。请读者用自己就读高校的校园风景图替换“南京师范大学仙林校区大门”的图像文件。

(1) 设计网页，文件名为 T2_xy.html。该网页介绍本校的基本情况，要求尽可能多地使用 XHTML 的标记。设计完成后，在浏览器上显示该网页。

(2) 修改文件 my_college.html，实现单击本学校的图像时，即可链接到 T2_xy.html 网页，将修改后的文件另存为 my_college_link.html。



图 1.26 南京师范大学校门背景网页

(3) 在浏览器上显示 my_college_link.html 网页，观察是否可以实现超链接功能。

2. 设计表格，文件名为 my_table.html，如图 1.27 所示（与图 1.10 相同）。

(1) 修改文件 my_table.html，使“通信工程”专业增加一个学生。

(2) 增加 2 门“英语”课程成绩，标题分为 2 行，第 1 行为“英语”，第 2 行分为 2 列，第 1 列内容为“(1)”，第 2 列内容为“(2)”。

3. 设计表单，文件名为 my_form.html，如图 1.28 所示（与图 1.18 相同）。

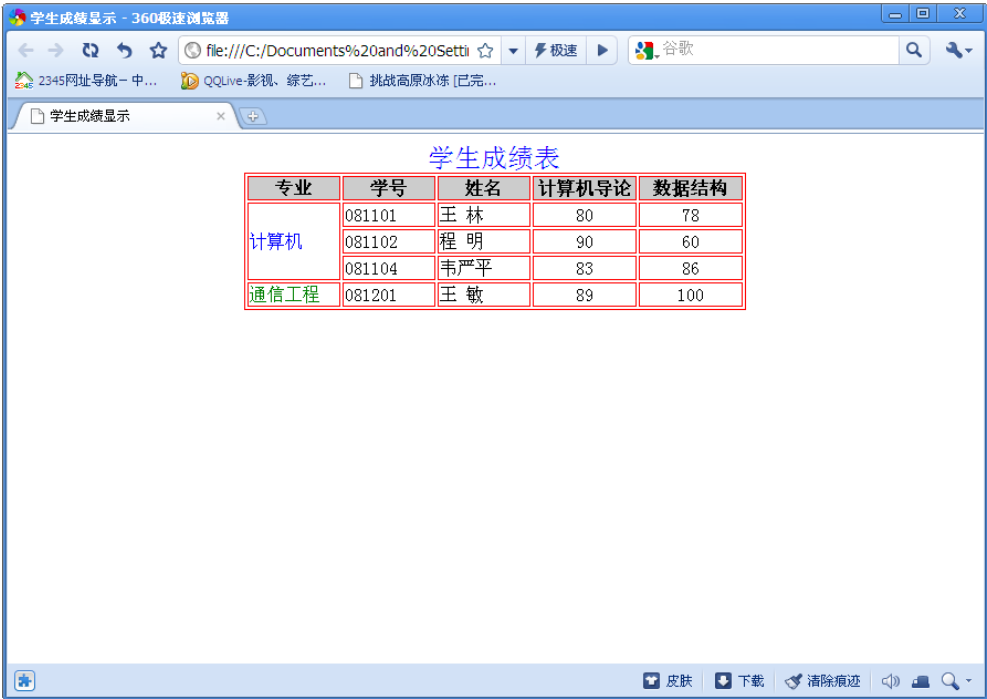


图 1.27 表格运行结果



图 1.28 表单运行结果

修改文件 my_form.html，在与“专业”同一列的位置增加“总学分”。

4. 设计框架，文件名为 my_frame.html，如图 1.29 所示（与图 1.20 相同）。

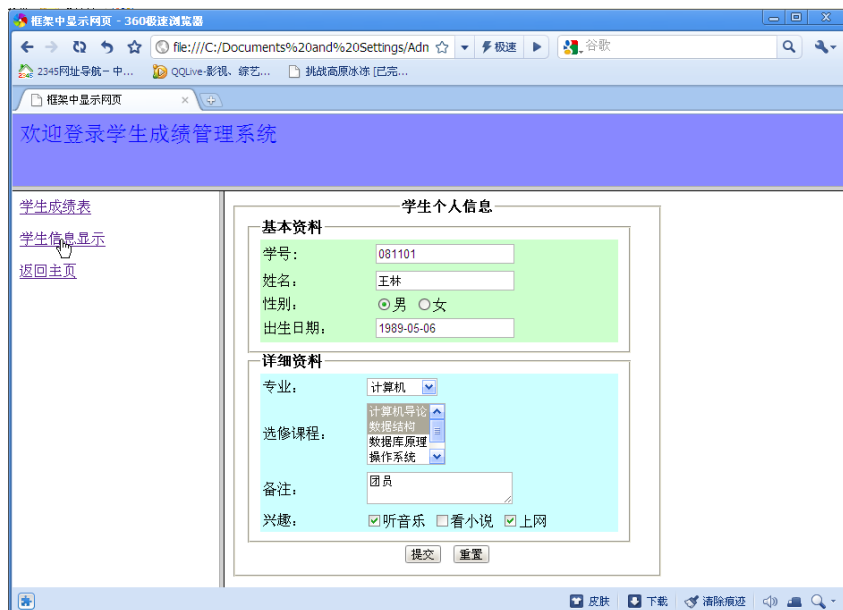


图 1.29 框架网页

(1) 将左边的超链接移到网页上部。左边网页设计为介绍“学生成绩管理系统”，系统的特点是在一个区域中从下向上滚动显示。

(2) 将左右窗口中显示的内容互换，将上下窗口位置互换。

第 2 章

JSP 入门

JSP (Java Server Pages) 是由原 Sun Microsystems 公司倡导、许多公司参与建立的一种动态网页技术标准。它在传统网页文件 (*.htm、*.html) 中插入 Java 程序段 (Scriptlet) 和 JSP 标记 (tag), 从而形成 JSP 文件 (*.jsp)。用 JSP 开发的 Web 应用是跨平台的, 既能在 Windows 下运行, 也能在 Linux/UNIX 下运行。

2.1 安装 JSP 运行环境

2.1.1 JDK 7 的安装与配置

安装 Java 开发包 (Java 2 Software Development Kit, JDK) 是进行 JSP 开发的前提, 其最新版本为 JDK 7.2, 所以本书所有范例都是基于 JDK 7.2 开发的。软件可以在 Oracle 公司 (原 Sun 公司已被 Oracle 收购) 的官方网站下载到, 网址如下:

<http://www.oracle.com/technetwork/java/index.html>

其安装过程非常简单, 这里就不再赘述了, 本书安装的目录是 “C:\Program Files\Java\jdk1.7.0_02”。

通过设置系统环境变量, 告诉 Windows 操作系统 JDK 7.2 的安装位置。下面具体介绍设置环境变量的方法。

(1) 设置系统变量 JAVA_HOME。右击 “我的电脑” 图标, 选择 “属性” → “高级” → “环境变量” 菜单项, 弹出 “环境变量” 对话框, 如图 2.1 所示。在 “系统变量” 中单击 “新建” 按钮, 弹出 “新建系统变量” 对话框, 在 “变量名” 文本框中输入 “JAVA_HOME”, 在 “变量值” 文本框中输入 JDK 的安装路径 “C:\Program Files\Java\jdk1.7.0_02”, 如图 2.2 所示, 单击 “确定” 按钮完成配置。

(2) 设置系统变量 Path。在 “环境变量” 对话框的 “系统变量” 列表中找到变量名为 “Path” 的变量, 单击 “编辑” 按钮, 在后面添加路径 “;C:\Program Files\Java\jdk1.7.0_02\bin”, 如图 2.3 所示, 单击 “确定” 按钮完成配置。

至此, JDK 的安装与配置就完成了, 读者可以自己测试是否已配置成功。选择 “开始” → “运行” 菜单项, 输入 “cmd”, 进入 DOS 界面。在命令行输入 “java -version”, 如果配置成功就会出现 Java 的版本信息, 如图 2.4 所示。

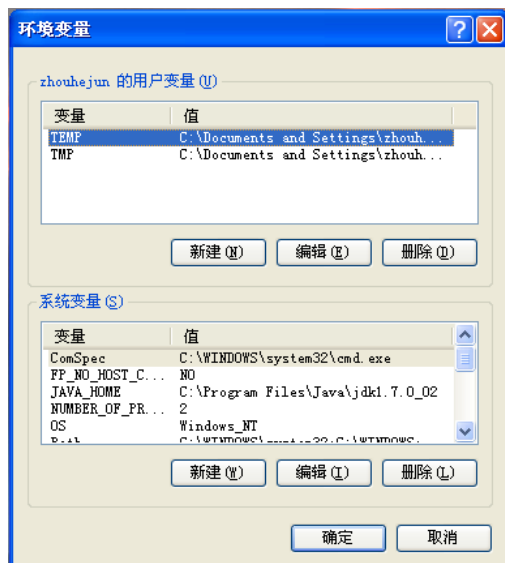


图 2.1 “环境变量”对话框

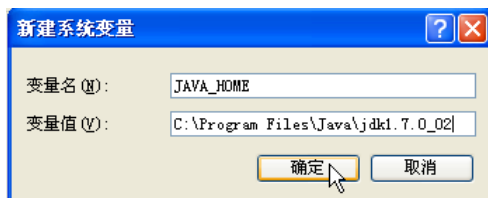


图 2.2 新建变量 JAVA_HOME

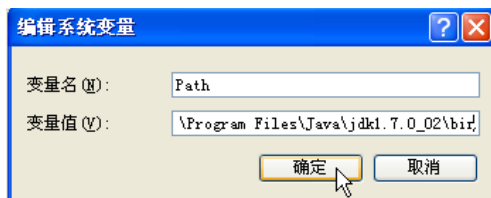


图 2.3 编辑变量 Path

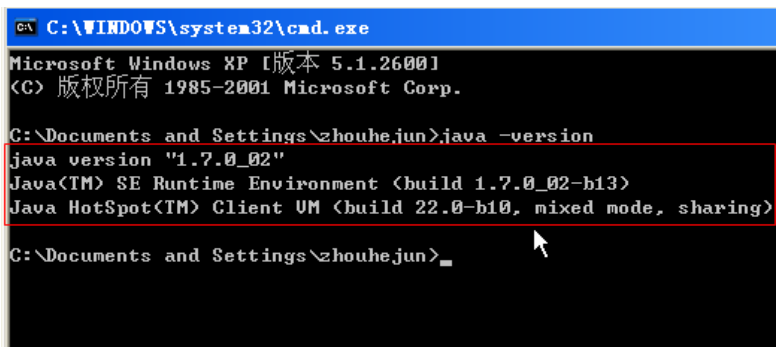


图 2.4 JDK 配置成功

2.1.2 Tomcat 7 的安装与配置

Tomcat 是一个免费开源的 Servlet 容器，它是 Apache 基金会资助 Jakarta 的一个核心子项目。本书采用最新的 Tomcat 7.0 作为 JSP 程序运行的 Web 服务器。Tomcat 7.0 可以在其官方网站 <http://tomcat.apache.org> 下载，本书出版社网站华信教育资源网 (www.hxedu.com.cn) 也提供了安装文件。

下面具体介绍其安装过程。

(1) 运行 apache-tomcat-7.0.23.exe，开始安装。

(2) 选择安装内容, 如图 2.5 所示, 这里采用默认内容。

(3) 设定连接端口 (HTTP/1.1 Connector Port)、登录名 (User Name) 和密码 (Password), 也采用系统默认值。连接端口为 8080, 登录名和密码均为空, 如图 2.6 所示。

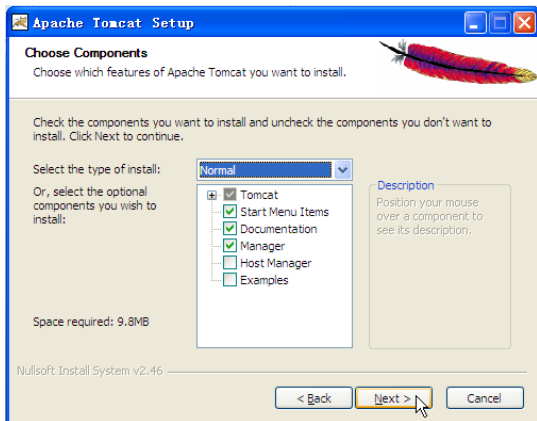


图 2.5 选择安装内容

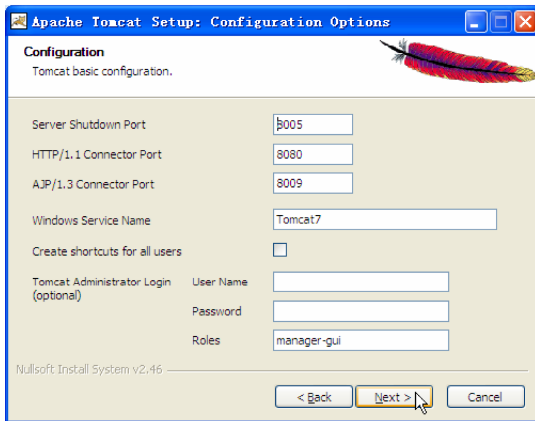


图 2.6 设定连接端口、登录名和密码

(4) 设定 Tomcat 使用的 JVM 路径 “C:\Program Files\Java\jre7”, 如图 2.7 所示。

(5) 选择安装路径, 默认为 “C:\Program Files\Apache Software Foundation\Tomcat 7.0”, 如图 2.8 所示, 单击 “Install” 按钮开始安装。

(6) 测试是否安装成功。安装完成后启动 Tomcat, 如图 2.9 所示。

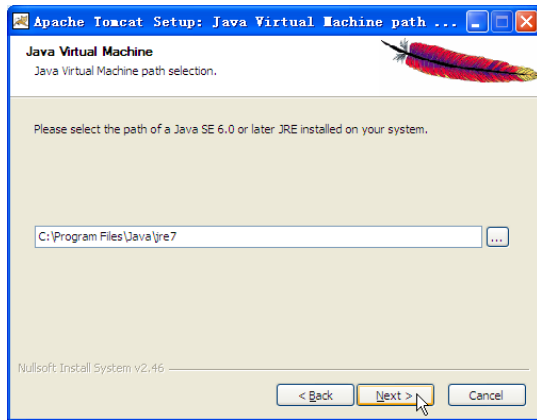


图 2.7 设定 Tomcat 使用的 JVM 路径

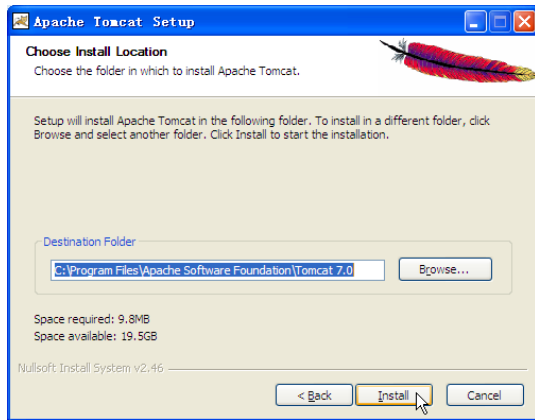


图 2.8 选择安装路径

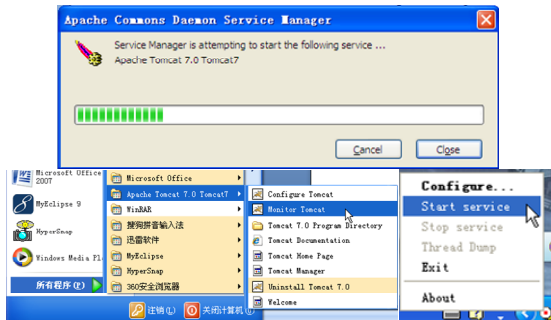


图 2.9 完成安装并启动 Tomcat

打开浏览器，输入“http://localhost:8080”，若如图 2.10 所示，表明安装成功。

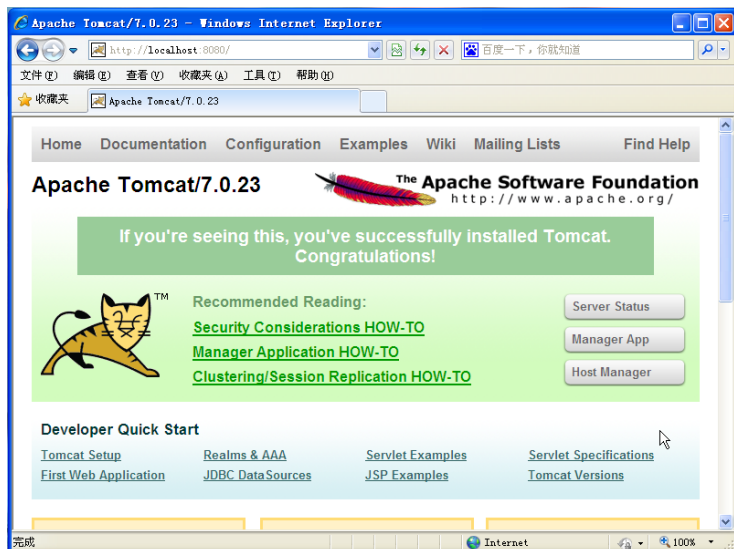


图 2.10 安装成功

2.1.3 MyEclipse 9.0 介绍

MyEclipse 企业级工作平台 (MyEclipse Enterprise Workbench, MyEclipse) 是对 Eclipse IDE 的扩展，作为一个极其优秀的用于开发 Java 应用的 Eclipse 插件集合，其功能非常强大，支持也很广泛，尤其是对各种开源产品的支持十分不错。利用它可以在数据库和 Java 应用的开发、发布，以及应用程序服务器的整合方面极大地提高工作效率。它是功能丰富的 Java 集成开发环境 (IDE)，包括了完备的编码、调试、测试和发布功能，完整支持 HTML/XHTML、JSP、CSS、Javascript、SQL 等各种 Java 相关技术的标准和框架。

本书用到了 MyEclipse 的最新稳定版本 MyEclipse 9.0，图 2.11 所示是它的启动画面和版本信息，至于安装过程，非常简单，跟着向导的步骤走就行了，不再介绍。

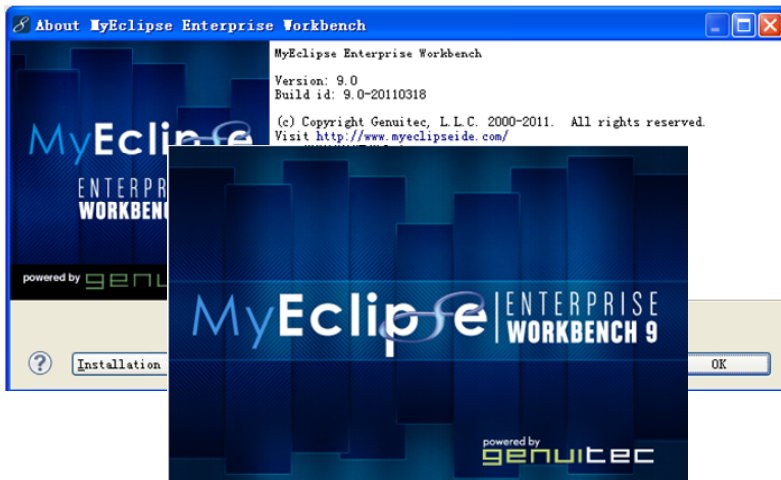


图 2.11 MyEclipse 9.0

MyEclipse 各版软件的破解历来都是 Java 初学者面临的重大难题之一!虽然 MyEclipse 至今已推出到 10.0 版,但只有 6.5GA、8.5 和 8.6.1 等少数几个是稳定版并有成熟的免注册安装文件。令人欣慰的是,经高手们的不懈努力,MyEclipse 9.0 终于也被成功破解了!根据网上的资料和亲身测试,笔者已将注册破解的详细操作步骤写成文档指南和完整的 MyEclipse 9.0 安装文件 (myeclipse-9.0-offline-installer-windows.exe) 在华信教育资源网上一起提供,希望此举能最大限度地帮助到众多 JSP 初学者。

在 Windows 下选择“开始”→“所有程序”→“MyEclipse”→“MyEclipse 9”菜单项,启动 MyEclipse 9.0 环境,进入集成开发工作界面,如图 2.12 所示。

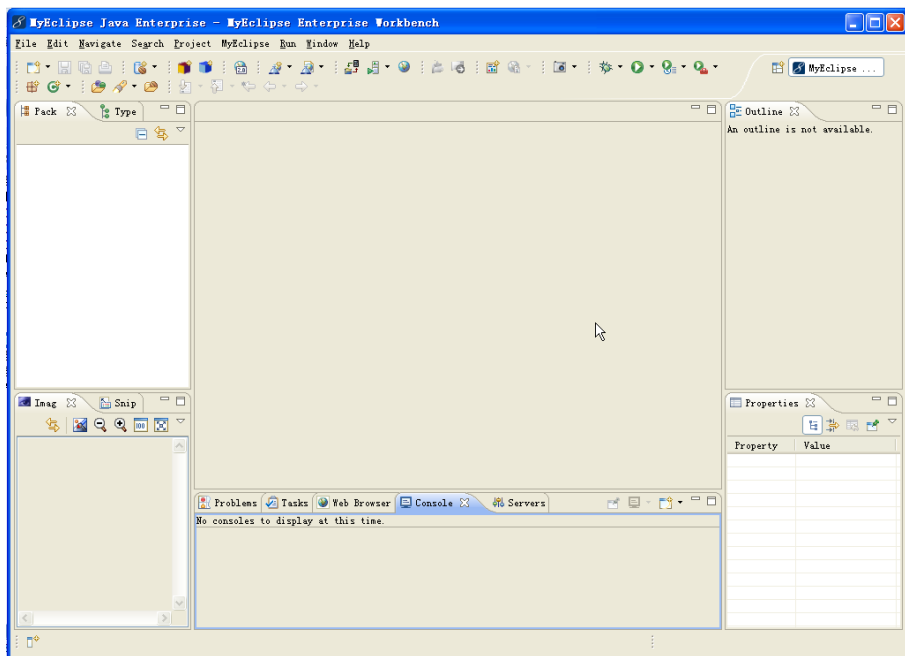


图 2.12 MyEclipse 9.0 主界面

在结构上,MyEclipse 的功能可以被分为 7 类:JavaEE 模型、Web 开发工具、EJB 开发工具、应用程序服务器的连接器、JavaEE 项目部署服务、数据库服务和 MyEclipse 整合帮助。

对于以上每个类别,在 MyEclipse 中都有相应的功能部件。MyEclipse 结构上的这种模块化,便于用户在不影响其他模块的情况下,对任意一个模块进行单独的扩展和升级。

事实上,MyEclipse 原本 (6.0 版之前) 只是作为 Eclipse 环境的一个插件而存在的,只不过后来随着它功能的日益强大,逐步取代 Eclipse 而成为独立的 JavaEE 集成开发环境,但在其界面主菜单里至今仍保留着 MyEclipse 的菜单项,如图 2.13 所示。

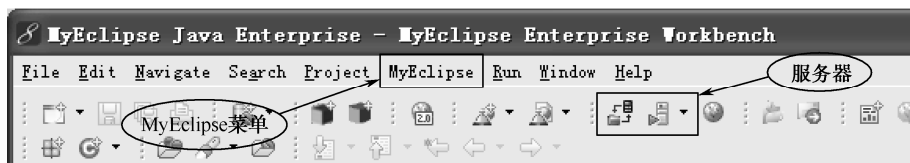


图 2.13 MyEclipse 菜单项

图 2.13 中还标示出了服务器部署、启动按钮，这都是开发时常用的，通过该功能可将项目部署到指定的软件服务器上。

MyEclipse 环境的这种功能组件化的集成定制特性，使得它可以用于开发多种不同类型的 Java 应用软件，如图 2.14 所示，在“File”→“New”菜单项下可以一览 MyEclipse 支持创建的所有项目类型和源文件类型（图 2.14 中框出的），还真不少！

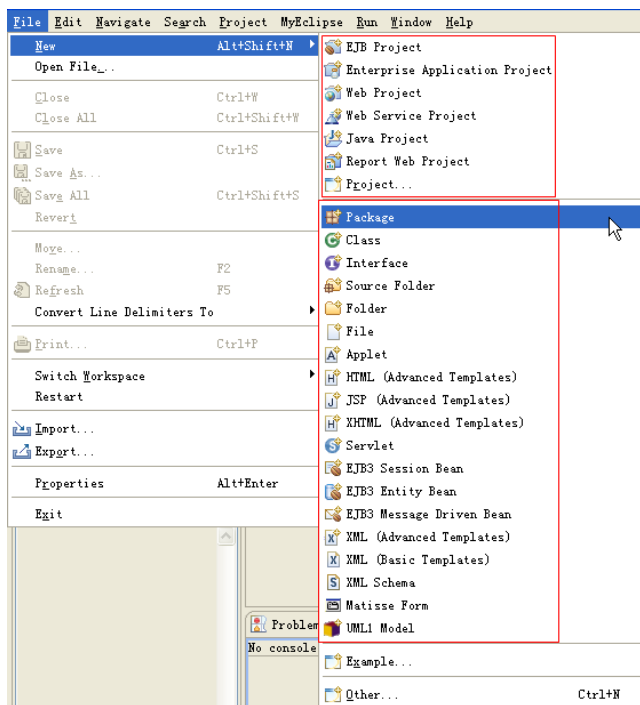


图 2.14 MyEclipse 支持创建的项目和源文件种类繁多

Genuitec 总裁 Maher Masri 曾说：“今天，MyEclipse 已经提供了意料之外的价值。其中的每个功能在市场上单独的价格都比 MyEclipse 要高。”

其实，开发 JSP 程序只要有 **JDK 和 Tomcat 就足够了**，但对于一些规模较大、比较复杂的 JSP 应用，使用 MyEclipse 工具快速构建项目能极大地提高开发效率。本章 2.5 节将引领大家体验用 MyEclipse 开发 JSP 的乐趣。

2.2 JSP 软件工作原理

JSP 技术是建立在 Java Servlet 之上的，想要真正了解 JSP 软件的工作原理，就必须先了解 Servlet 机制。

2.2.1 Servlet 基础

1. Servlet 的概念

Servlet 是一种服务器端的 Java 程序，具有独立于平台和协议的特性，可以生成动态的 Web 页面。Servlet 由 Web 服务器进行加载，而该 Web 服务器必须包含支持 Servlet 的 JVM（Java

虚拟机)。

Servlet 是位于 Web 服务器**内部**的服务器端的 Java 应用程序,它担当客户 (Web 浏览器) 请求与服务器 (Web 服务器上的应用程序) 响应的中间层,基于这种“请求/响应”模型,Servlet 模块的运行模式如图 2.15 所示。

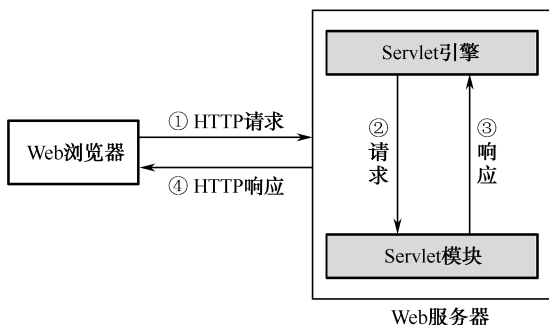


图 2.15 Servlet 运行模式

从图 2.15 中可以看出,整个处理流程如下。

① HTTP 请求。Web 浏览器将客户请求发送给 Web 服务器上的 Servlet 引擎。

② 请求。Servlet 引擎将请求转发给处理请求的 Servlet 模块。

③ 响应。Servlet 模块接受请求后,调用相应的服务 (service()) 对请求进行处理,然后将处理结果返给 Servlet 引擎。

④ HTTP 响应。Servlet 引擎将结果发送给客户端。

2. Servlet 的基本结构

Servlet 模块是用 Servlet API 编写的,Servlet API 包含两个包: javax.servlet 和 javax.servlet.http。

图 2.16 清晰地描绘了这两个包中主要类、接口之间的关系。

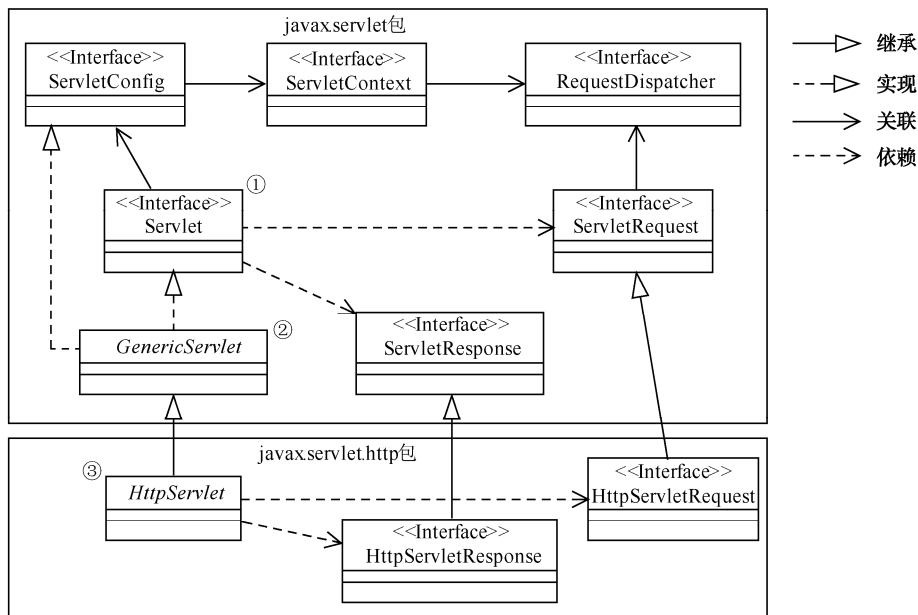


图 2.16 Servlet API 的类框图

其中, javax.servlet 包中的类与 HTTP 协议无关; javax.servlet.http 包中的类与 HTTP 协议相关, 该包中的部分类继承了 javax.servlet 包中的部分类和接口。

这两个包是 Servlet API 的基本包, 它们都封装在文件 servlet-api.jar 中, 而文件 servlet-api.jar 在 Tomcat 的 lib 目录下 (见图 2.17)。为了方便接下来编写 Servlet 程序, 需要读者先创建一个环境变量 CLASSPATH (设置环境变量的方法见 2.1.1 节), 然后在该变量中添加类路径 “;C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar”, 如图 2.18 所示。

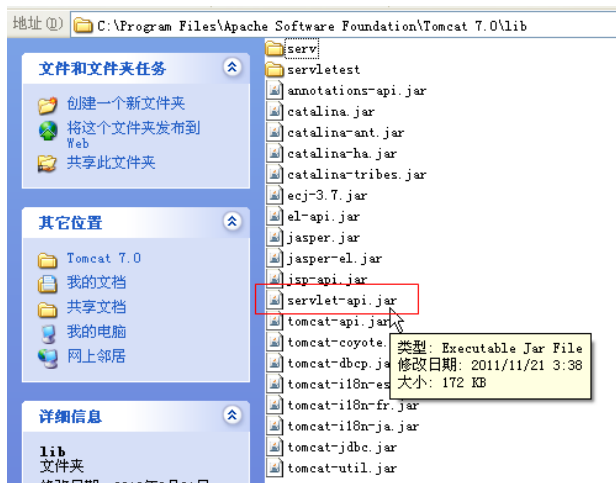


图 2.17 Servlet API 所在类库.jar 包

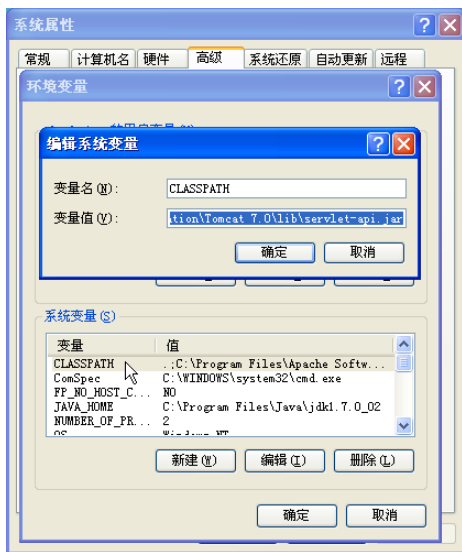


图 2.18 设置环境变量 CLASSPATH

3. Servlet 的编程方式

为了让大家更容易理解 Java 语言中 Servlet 的基本结构和使用方法, 下面先以几个最简单的小程序为例, 比照图 2.16, 循序渐进、逐层深入地介绍 Servlet 编程的几种基本方式。

第一种: 实现 Servlet 接口的方式。

Servlet 程序不是独立的应用程序, 它没有 main() 方法, 而是生存在容器中, 由容器来管理。编写一个 Servlet 程序, 需要实现 javax.servlet.Servlet 接口 (见图 2.16 中的①)。

【例 2.1】用实现 Servlet 接口的方式编写一个 Servlet 程序。

(1) 创建用户目录

编写 Servlet 类之前, 首先创建一个用户目录, 用以保存 Servlet 源文件。这里创建一个目录 “D:\ch_2”。

(2) 编写自己的 Servlet 类

用记事本工具编写一个 Servlet 类, 如下:

```
package servletest;
import java.io.*;
import javax.servlet.*;
public class _2_1hello implements Servlet {
    public void init(ServletConfig arg0) throws ServletException {
    } //①init()方法
    public ServletConfig getServletConfig() {
```

```
        return null;
    }           //②getServletConfig()方法
    public String getServletInfo() {
        return null;
    }           //③getServletInfo()方法
    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException {
        PrintWriter out=response.getWriter();
        out.println("<html><body>");
        out.println("<font size=6 color=red>Hello World!</font>");
        out.println("</body></html>");
    }           //④service()方法
    public void destroy() {
    }           //⑤destroy()方法
}
```

编辑完成后，以文件名 `_2_1hello.java` 保存在“D:\ch_2”下，编译生成 `_2_1hello.class` 字节码文件。

从上面这个典型的 `Servlet` 类的结构可以看出，它实现了 `Servlet` 接口，即实现了 `Servlet` 接口定义的五五个方法。

下面介绍这五个方法的作用。

① `init()`方法。在 `Servlet` 实例化之后，`Servlet` 容器会调用 `init()`方法，来初始化该对象。该方法主要是为了让 `Servlet` 对象在处理客户请求前可以完成一些初始化的工作，比如建立数据库的连接，获取配置信息等。对于每个 `Servlet` 实例，`init()`方法只能被调用一次。`init()`方法有一个类型为 `ServletConfig` 的参数，`Servlet` 容器通过这个参数向 `Servlet` 传递配置信息。`Servlet` 使用该对象从 Web 应用程序的配置信息中获取初始化参数。另外，在 `Servlet` 中，还可以通过 `ServletConfig` 对象获取描述 `Servlet` 运行环境的 `ServletContext` 对象，使用该对象，`Servlet` 可以和它的 `Servlet` 容器进行通信。

② `getServletConfig()`方法返回容器调用 `init()`方法时传递给 `Servlet` 对象的 `ServletConfig` 对象。`ServletConfig` 对象包含了 `Servlet` 的初始化参数。

③ `getServletInfo()`方法返回一个 `String` 类型的字符串，其中包括了关于 `Servlet` 的信息，如作者、版本和版权。

④ `service()`方法。容器调用 `service()`方法来处理客户端的请求。在 `init()`方法正确完成后，`service()`方法被容器调用。在 `service()`方法中有一个用于接收客户端请求信息的请求对象（类型为 `ServletRequest`）和一个用于对客户端进行响应的响应对象（类型为 `ServletResponse`）的参数。`Servlet` 对象通过 `ServletRequest` 对象得到客户端的相关信息和请求信息，在对请求进行处理后，调用 `ServletResponse` 对象的方法设置响应信息。

⑤ `destroy()`方法。当容器检测到一个 `Servlet` 对象应该从服务中被移除的时候，容器会调用该对象的 `destroy()`方法，来释放 `Servlet` 对象所使用的资源，保存数据到持久存储设备中。例如，将内存中的数据保存到数据库中，关闭数据库的连接等。在 `Servlet` 容器调用 `destroy()`方法前，如果还有其他线程正在 `service()`方法中执行，容器会等待这些线程执行完毕或等待服务器设定的最大时间到达。一旦 `Servlet` 对象的 `destroy()`方法被调用，容器便不会再把其他的请求发送给该对象。如果需要该 `Servlet` 再次为客户端服务，容器将会重新产生一个 `Servlet` 对

象来处理客户端的请求。在 `destroy()` 方法调用之后，容器会释放这个 `Servlet` 对象，在随后的时间内，该对象会被 Java 的垃圾收集器所回收。

(3) 部署 Servlet

Tomcat 服务器存放 Servlet 字节码文件的主目录是 `C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib`，在该目录下建立一个子目录 `servletest`，将刚刚生成的字节码文件 `_2_1hello.class` 复制到这个子目录下。

在 `C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ROOT\WEB-INF` 目录下，打开 `web.xml` 文件，将光标移到文件末尾，在 `</web-app>` 标记前插入以下（加黑背景部分的）代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
...
-->
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0"
  metadata-complete="true">
  <display-name>Welcome to Tomcat</display-name>
  <description>
    Welcome to Tomcat
  </description>
  <servlet>
    <servlet-name>hello_1</servlet-name>
    <servlet-class>servletest._2_1hello</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hello_1</servlet-name>
    <url-pattern>/myserv1</url-pattern>
  </servlet-mapping>
</web-app>
```

下面介绍一下这段 `web.xml` 的配置信息。

第一行是对 xml 文件的声明，然后是 xml 的根元素 `<web-app>`，其属性中声明了版本等信息，这是固定的头文件，Tomcat 早已填写好了，读者无须改动。

接着（上述代码中加黑背景）的部分才是需要配置的内容。

`<servlet>` 与 `</servlet>` 之间配置的是 `<servlet-name>` 和 `<servlet-class>`。其中 `<servlet-name>` 的值 `hello_1` 是程序员自己为 `servlet` 起的一个名字（起名需要符合 Java 的命名规则）；而 `<servlet-class>` 的值则是前面编写的 `Servlet` 类的类名，这个必须配置正确，如果有包，还要在前面加上包名，如本例为 `servletest._2_1hello`（注意，这里的类名不带 `.java`，也不带 `.class`）。

`<servlet-mapping>` 与 `</servlet-mapping>` 之间配置的是 `<servlet-name>` 与 `<url-pattern>`。其中 `<servlet-name>` 的值就是上面刚刚配置的 `<servlet-name>` 的值，而 `<url-pattern>` 的值也可以随便起名，但其前面必须加 `“/”`，如本例写为 `/myserv1`，是该 `Servlet` 运行的路径名。

(4) 运行 Servlet

启动 Tomcat 7.x, 在浏览器中输入 “http://localhost:8080/ myserv1”, 就会在页面中显示 “Hello World!”, 如图 2.19 所示。



图 2.19 【例 2.1】运行界面

此处 “http://localhost:8080/” 是服务器 URL, 后面紧跟的 “myserv1” 就是在 web.xml 文件中配置的 <url-pattern> 的值。

第二种：继承 GenericServlet 类的方式。

在【例 2.1】中采用实现 Servlet 接口的方式定义了一个 Servlet 类, 这样做需要实现 Servlet 接口的 5 个方法。为了简化 Servlet 的编写, javax.servlet 包提供了一个抽象的类 GenericServlet (见图 2.16 中的②)。它给出了除 service() 方法外的其他 4 个方法的简单实现, 并且还实现了 ServletConfig 接口, 如果在编程中直接继承这个类, 代码会简化很多。

【例 2.2】用继承 GenericServlet 类的方式编写一个 Servlet 程序。

源文件位于 D:\ch_2 下, 文件名为 _2_2hello.java, 代码如下:

```
package servletest;
import java.io.*;
import javax.servlet.*;
public class _2_2hello extends GenericServlet{
    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException {
        PrintWriter out=response.getWriter();
        out.println("<html><body>");
        out.println("<font size=6 color=red>Hello World!</font>");
        out.println("</body></html>");
    }
}
```

编译后的字节码文件同样复制到子目录 servletest 下, 并在 web.xml 文件中进行配置:

```
<servlet>
    <servlet-name>hello_2</servlet-name>
```

```

        <servlet-class>servletest._2_2hello</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>hello_2</servlet-name>
        <url-pattern>/myserv2</url-pattern>
    </servlet-mapping>

```

运行后一样可以输出“Hello World!”。

第三种：继承 `HttpServlet`、覆盖 `doXXX()` 的方式。

在大部分网络中，都是客户端通过 `http` 协议来访问服务器端的资源。为了快速开发应用于 `http` 协议的 `Servlet` 类，在 `javax.servlet.http` 包中提供了一个抽象类 `HttpServlet`（见图 2.16 中的③），它继承了 `GenericServlet` 类。

在 `HttpServlet` 类中重载了 `GenericServlet` 的 `service()` 方法：

- ① `public void service(ServletRequest request, ServletResponse response)`
throws `ServletException`, `java.io.IOException`
- ② `protected void service(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`

根据不同的请求方法，`HttpServlet` 提供了七个处理方法：

- ① `protected void doGet(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`
- ② `protected void doPost(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`
- ③ `protected void doHead(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`
- ④ `protected void doPut(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`
- ⑤ `protected void delete(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`
- ⑥ `protected void doTrace(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`
- ⑦ `protected void doOptions(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`

当容器接收到一个针对 `HttpServlet` 对象的请求时，该对象就会调用 `public` 的 `service()` 方法，首先将参数类型转换为 `HttpServletRequest` 和 `HttpServletResponse`，然后调用 `protected` 的 `service()` 方法将参数送进去，接着调用 `HttpServletRequest` 对象的 `getMethod()` 方法获取请求方法名来调用相应的 `doXXX()` 方法，所以一个 `Servlet` 类在继承 `HttpServlet` 的时候，不一定要覆盖它的 `service()` 方法，只需要覆盖相应的 `doXXX()` 方法就行了。通常情况下，都是覆盖其 `doGet()` 和 `doPost()` 方法，然后在其中的一个方法里调用另一个方法，做到合二为一。

【例 2.3】 用继承 `HttpServlet`、覆盖 `doGet()` 和 `doPost()` 的方式编写一个 `Servlet` 程序，实现与【例 2.1】和【例 2.2】一样的输出“Hello World!”的功能。

源文件位于 `D:\ch_2` 下，文件名为 `_2_3hello.java`，代码如下：

```

package servletest;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

```

```

public class _2_3hello extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out=response.getWriter();
        out.println("<html><body>");
        out.println("<font size=6 color=red>Hello World!</font>");
        out.println("</body></html>");
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request,response);
    }
}

```

编译后的字节码文件同样复制到子目录 `servletest` 下，并在 `web.xml` 文件中进行配置：

```

<servlet>
    <servlet-name>hello_3</servlet-name>
    <servlet-class>servletest._2_3hello</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>hello_3</servlet-name>
    <url-pattern>/myserv3</url-pattern>
</servlet-mapping>

```

运行后输出的也是“Hello World!”。

第四种：继承 `HttpServlet`、重写 `service()` 的方式。

继承 `HttpServlet` 编写 `Servlet` 程序，是目前最常用的方式，其本质就是扩展 `HttpServlet` 类，因此原则上可以将要处理的逻辑代码写在 `service()`、`doXXX()` 系列方法的任意一个中（一般只写在 `service()`、`doPost()` 或 `doGet()` 这三个常用方法中）。

为简单起见，用户只需重写 `service()` 方法，`Servlet` 模块执行 `service()` 方法时，会自动调用 `doPost()` 和 `doGet()` 这两个方法，实现 `Servlet` 的逻辑处理功能。

【例 2.4】用继承 `HttpServlet`、重写 `service()` 的方式实现一个 `Servlet` 程序，要求以中英文两种形式分别输出“Hello World!”及“你好，世界!”。

源文件位于 `D:\ch_2` 下，文件名为 `_2_4hello.java`，代码如下：

```

package servletest;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class _2_4hello extends HttpServlet{
    public void init(ServletConfig config)throws ServletException
    {
        super.init(config);
    }
    public void service(HttpServletRequest request,HttpServletResponse response)throws IOException
    {
        response.setContentType("text/html;charset=GB2312");//为了输出中文，要设置汉字编码
    }
}

```



```
        PrintWriter out=response.getWriter();
        out.println("<html><body>");
        out.println("<font size=6 color=red>Hello World!你好，世界！ </font>");
        out.println("</body></html>");
    }
}
```

编译后的字节码文件同样复制到子目录 `servletest` 下，并在 `web.xml` 文件中进行配置：

```
<servlet>
    <servlet-name>hello_4</servlet-name>
    <servlet-class>servletest._2_4hello</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>hello_4</servlet-name>
    <url-pattern>/myserv4</url-pattern>
</servlet-mapping>
```

运行后的输出结果如图 2.20 所示。



图 2.20 【例 2.4】运行界面

4. Servlet 生命周期

当服务器调用 `Servlet` 类时，`Servlet` 对象被创建。从服务器创建 `Servlet` 对象到该对象被消灭这段时间称为 `Servlet` 生命周期。

当 `Servlet` 被装载到容器后，生命周期开始。首先调用 `init()` 方法进行初始化，初始化后，调用 `service()` 方法，根据请求的不同调用不同的 `doXXX()` 方法处理客户请求，并将处理结果封装到 `HttpServletResponse` 中返给客户端。当 `Servlet` 对象从容器中移除时调用其 `destroy()` 方法，这就是 `Servlet` 运行的整个过程，可以据此画出 `Servlet` 对象的执行流程图，如图 2.21 所示。

5. Servlet 程序示例

下面综合运用前面所讲（包括第 1 章）的知识，编写一个基于浏览器运行的 `Java Servlet` 小程序，以展示 `Servlet` 的常规用途。

是存放于互联网上某台服务器中的，原理是一样的)。

(2) 编写后台 Servlet 类

源文件位于 D:\ch_2 下，文件名为 _2_5area.java，代码如下：

```
package servlettest;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class _2_5area extends HttpServlet{
    public void init(ServletConfig config)throws ServletException
    {
        super.init(config);
    }
    public void service(HttpServletRequest request,HttpServletResponse response)throws IOException
    {
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();
        out.println("<html><body>");
        double r,s;
        String radius=request.getParameter("radius");
        if(radius==null){s=0.0;}
        else{
            r=Double.parseDouble(radius);
            s=3.14*r*r;
        }
        out.println("<font size=\"5\" color=\"blue\">圆面积计算</font><br>");
        out.println("<t<form action=\"/getResult\" method=\"post\">");
        out.print("<t 请输入半径 r: <input type=\"text\" name=\"radius\" value=\"\"");
        out.print(radius);
        out.println("<\" size=\"4\">&nbsp;&nbsp;&nbsp;");
        out.println("<t<t<t<input type=\"submit\" value=\"计算\"><br><br>\r\n");
        out.print("<t 面积 S 为: <input type=\"text\" name=\"area\" value=\"\"");
        out.print(s);
        out.println("<\" size=\"7\">");
        out.println("<t</form>");
        out.println("</body></html>");
    }
}
```

编译上段代码，生成 _2_5area.class 字节码文件。

(3) 部署 Servlet

将 _2_5area.class 字节码文件复制到子目录 servlettest 下，并在 web.xml 文件中进行配置：

```
<servlet>
    <servlet-name>area_5</servlet-name>
    <servlet-class>servlettest._2_5area</servlet-class>
</servlet>
<servlet-mapping>
```

```
<servlet-name>area_5</servlet-name>
<url-pattern>/getResult</url-pattern>
</servlet-mapping>
```

此处配置<url-pattern>为“/getResult”，是与 XHTML 页面中的

```
<form action="/getResult" method="post">
```

相对应的，这里指明了处理提交表单的 Servlet 所在的服务器路径就是/getResult。

(4) 运行程序

启动 Tomcat，打开 IE 浏览器，在地址栏输入“http://localhost:8080/2_5area.html”后按回车键，在界面中填写圆半径为“10”，单击“计算”按钮，其运行结果如图 2.22 所示。

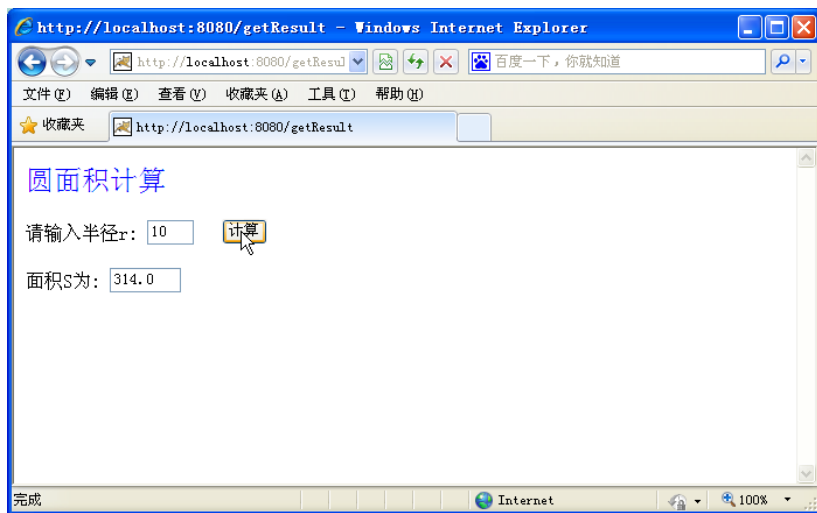


图 2.22 用 Servlet 实现圆面积计算

2.2.2 JSP 软件体系结构

1. Servlet 编程的缺陷

仔细观察【例 2.5】程序_2_5area.java 源文件中的代码，大家就会发现：Servlet 不仅要负责计算圆面积，而且还要将计算结果嵌在页面中，再用形如

```
out.println("...");
out.println("...");
...
```

的语句一行一行地发送给客户端。事实上，每次计算完毕后，整个客户端页面都要由 Servlet 重新生成一遍！这个 Servlet 不仅要完成计算圆面积的本职工作，还肩负着一次次重新生成客户端页面的重任，真辛苦啊！

不能将页面显示的代码和逻辑处理的代码有效地隔离，这是 Servlet 编程的重大缺陷。

2. JSP 原理

为了克服 Servlet 的这一致命缺陷，人们发明了 JSP。JSP (Java Server Pages) 是一种实现普通静态 XHTML 和动态 XHTML 混合编码的技术，它并没有增加任何本质上不能用 Servlet 实现的功能。但是，在 JSP 中编写静态 XHTML 更加方便，不必再用 println 语句来输出每一行 XHTML 代码。更重要的是，借助内容和外观的分离，页面制作中不同性质的任务可以方

便地分开，比如：由页面设计者进行 XHTML 设计，同时留出供 Servlet 程序员插入动态内容的空间。

要实现这一目标，只需在 Servlet 体系中引入一个 JSP 引擎，其结构如图 2.23 所示。

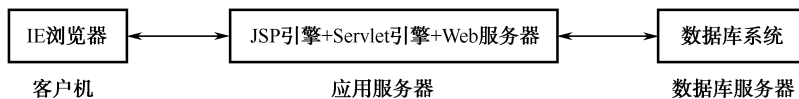


图 2.23 JSP 软件体系

JSP 引擎统一整合 XHTML 和 Java 代码来处理动态页面：每页第一次被调用时，都通过 JSP 引擎自动地被编译成 Servlet 程序，然后才被执行，其工作流程如图 2.24 所示。

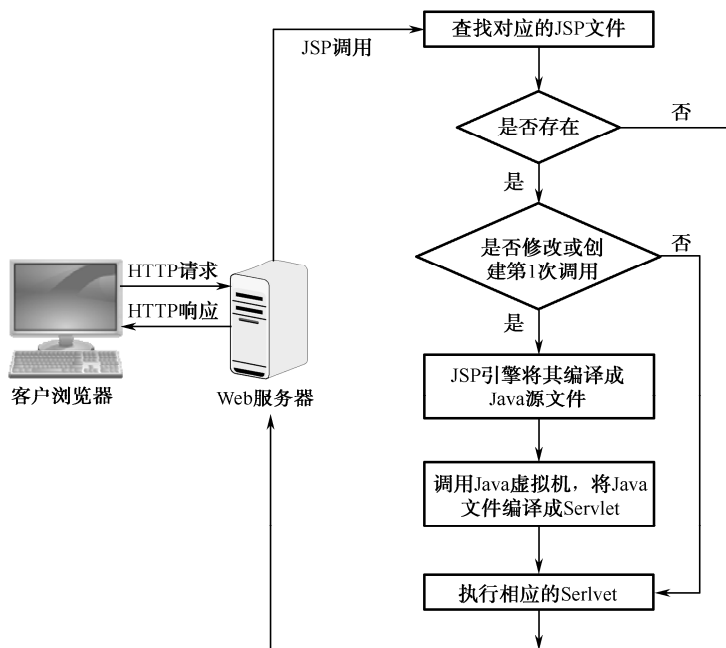


图 2.24 JSP 工作流程图

从图 2.24 所示的流程可知，当一个 JSP 文件第一次被请求时，JSP 引擎先把该 JSP 文件转换成一个 Java 源文件，在转换时，如果发现 JSP 文件有任何语法错误，转换过程将中断，并向服务器端和客户端输出错误信息；如果转换成功，JSP 引擎调用 Java 虚拟机的 `javac` 程序把该 Java 文件的源文件编译成相应的 `.class` 文件，该 `.class` 文件也就是一个 Servlet 程序，然后创建一个该 Servlet 的实例，提供服务响应用户的请求。

3. JSP 代码构成

由图 2.23 可见，JSP 技术开发的程序结构只能是 B/S/S 结构或 B/S 结构。通常情况下，JSP 程序由展示用户界面的 XHTML 标记和进行数据计算的 Java 代码两部分组成，因此数据展示层代码和数据计算代码可能包含在同一个 JSP 文件中，它们都部署在应用服务器上。

JSP 源代码可能有以下三种形式。

- ① JSP 源代码=XHTML 标记+Java 程序片。
- ② JSP 源代码=XHTML 标记+Servlet 模块+Java 程序片。
- ③ JSP 源代码=XHTML 标记+Bean+Java 程序片。

一般来说, Java 程序片、Servlet 模块负责实现逻辑计算功能; Bean 负责实现数据处理功能; XHTML 标记负责实现数据展示功能。JSP 页面中的 Java 程序片最终被 JSP 引擎转译为 Servlet 模块, 当客户发出 Servlet 请求时, 再由 Servlet 引擎将这些应用 Servlet 模块载入内存运行, 以处理客户请求。

2.2.3 JSP 程序执行流程

一个 JSP 页面可以被多个客户访问, 下面是第一个客户访问 JSP 页面时, 页面的执行过程。

- ① 客户通过浏览器向服务器端的 JSP 页面发送请求。
- ② JSP 引擎检查 JSP 文件对应的 Servlet 源代码是否存在, 若不存在转向第④步, 否则执行下一步。
- ③ JSP 引擎检查 JSP 页面是否修改, 若未修改, 转向第⑤步, 否则执行下一步。
- ④ JSP 引擎将 JSP 页面文件转译为 Servlet 源代码 (相应的 .java 文件)。
- ⑤ JSP 引擎将 Servlet 源代码编译为相应的字节码 (.class 文件)。
- ⑥ JSP 引擎加载字节码到内存。
- ⑦ 字节码 (应用 Servlet) 处理客户请求, 并将结果返回给客户。

下面是 JSP 页面的执行流程, 如图 2.25 所示。

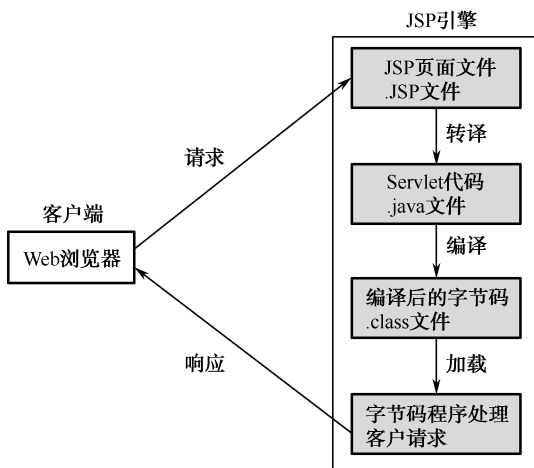


图 2.25 JSP 页面的执行流程

在不修改 JSP 页面的情况下, 除了第一个客户访问 JSP 页面需要经过以上几个步骤外, 以后访问该 JSP 页面的客户请求, 会被直接发送给 JSP 对应的字节码程序处理, 并将处理结果返给客户。在这种情况下, JSP 页面既不需转译也不需编译, 页面执行效率非常高。

2.3 一个简单的 JSP 例子

2.3.1 JSP 实现圆面积计算

在 2.2.1 节实现了一个用 Servlet 进行圆面积计算的程序, 本节改用 JSP 实现同样的功能,

读者不仅可以从中了解 JSP 代码的基本构成和编写规则,而且更能深刻体会到 JSP 较之 Servlet 编程的优势所在。

【例 2.6】用 JSP 重新实现【例 2.5】的那个计算圆面积的程序，功能要求和页面显示效果均不变。

(1) 用 Windows 的记事本输入 JSP 代码

输入以下内容，以 2_area.jsp 作为文件名保存。保存到的目录为 C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ROOT。

注意：JSP 文件保存时后缀名要用小写。

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<body>
    <%
        double r,s;
        String radius=request.getParameter("radius");
        if(radius==null){s=0.0;}
        else{
            r=Double.parseDouble(radius);
            s=3.14*r*r;
        }
    %>
    <font size="5" color="blue">圆面积计算</font><br>
    <form action="" method="post" name="fl">
        请输入半径 r:   <input type="text" name="radius" value="0" size="4">&nbsp;  &nbsp; 
                        <input type="submit" value="计算"><br><br>
        面积 S 为:       <input type="text" name="area" size="7">
    </form>
</body>
</html>
<script language="javascript">
    document.fl.area.value="<%=s%>";
    document.fl.radius.value="<%=radius%>";
</script>
```

(2) 在 Tomcat 服务器上执行 2_6area.jsp

在 IE 浏览器中输入“http://localhost:8080/2_6area.jsp”，系统显示 2_6area.jsp 的初始界面。输入半径为“10”后，单击“计算”按钮，其运行结果如图 2.26 所示。

用户的 JSP 文件一般保存到 Tomcat 服务器默认的 C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ROOT 目录中, 如果保存到 ROOT 下的其他目录, 则在 IE 浏览器中输入 http://localhost:8080/2_6area.jsp 时, 需要在 2_6area.jsp 文件前加入目录路径。

实际上，前面的系统 JSP 测试页面 index.jsp 和相关文件，就是在 Tomcat 服务器安装时存放到 C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ROOT 目录中的。



图 2.26 用 JSP 实现圆面积计算

2.3.2 JSP 程序基本构成

下面以【例 2.6】的 JSP 程序代码为例，介绍一般 JSP 程序的构成要素和编写规范，让读者对 JSP 程序设计有个初步的了解。

1. JSP 程序块

先来看【例 2.6】源文件中的这样一段 JSP 代码：

```
<%  
    double r,s;  
    String radius=request.getParameter("radius");  
    if(radius==null){s=0.0;}  
    else{  
        r=Double.parseDouble(radius);  
        s=3.14*r*r;  
    }  
%>
```

从上面的这段代码中可以发现，在<%与%>之间就是一个 Java 代码片段。这就是在 html 脚本中嵌入 Java 片段的方法，其中还可以定义数据类型，也就是说在<%与%>之间可以是任意的操作 Java 代码，这就为编写 JSP 文件带来了很大的方便。

2. JSP 数据定义

在 JSP 中可以用<%!和%>定义一个或多个变量。凡是在其中定义的变量即为该页面级别的共享变量，可以被访问此网页的所有用户访问，其语法格式如下：

```
<%! 变量声明 %>
```

例如，定义圆半径 r 和面积 S 的声明也可以写为如下形式：

```
<%! double r = 0.0,s;%>
```

此外，这种声明方式还可用于定义一个方法或类，定义方法的格式如下：

```
<%!  
    返回类型 方法名(参数类型 参数,...){
```



```

        语句;
        return (返回值);
    }
%>

```

定义一个类，如下面的代码片段：

```

<%!
    public class A {...}
%>

```

不过，这种方式不常用。

3. JSP 表达式

从【例 2.6】中可以发现，要输出面积 S 的值，应先计算 S 的值，然后输出结果。JSP 中提供了一种表达式，可以很方便地输出运算结果，其格式如下：

```

<%= Java 表达式 %>

```

于是上面的 JSP 程序块代码可以修改为：

```

<%@ page contentType="text/html; charset=GB2312" %>
<html>
<body>

```

```

    <%! double r=0.0,s;%>

```

```

    <%
        String radius=request.getParameter("radius");
        if(radius==null){s=0.0;}
        else{
            r=Double.parseDouble(radius);
        }
    %>

```

```

    ...

```

```

</body>
</html>
<script language="javascript">
    document.fl.area.value="<%=3.14*r*r%>";
    document.fl.radius.value="<%=radius%>";
</script>

```

可以输出同样的运算结果。

4. JSP 指令

JSP 指令主要用来提供整个 JSP 页面的相关信息和设定 JSP 页面的相关属性，如设定网页的编码方式、脚本语言及导入需要用到的包等，其语法格式如下：

```

<%@ 指令名 属性名="属性值"%>

```

常用的 JSP 指令有三条：page、include 和 taglib，如【例 2.6】源文件一开头就用到了 page 指令：

```

<%@ page contentType="text/html; charset=GB2312" %>

```

至于其他指令的作用，将在第 3 章的 3.3 节进行详细介绍。

以上结合【例 2.6】分别简要地介绍了 JSP 程序块、JSP 数据定义、JSP 表达式和 JSP 指令，它们都是构成 JSP 程序最基本的元素。此外，JSP 的基本元素还包括 JSP 动作和 JSP 注释（本例尚未涉及）。有关 JSP 动作的概念，读者将在稍后的 2.4 节接触到，而 JSP 注释则会在本

书后续章节的一些规模较大的 JSP 程序中经常用到，到时再进行讲解。

2.3.3 JSP 运行机制分析

1. _2_005f6area_jsp.java 文件

执行完【例 2.6】的源文件 2_6area.jsp 后，进入 C:\Program Files\Apache Software Foundation\Tomcat 7.0\work\Catalina\localhost_org\apache\jsp 目录下可以看到 2_005f6area_jsp.java 这个文件，这就是在刚才执行 2_6area.jsp 时经 JSP 引擎编译生成的 Servlet 文件，其代码如下：

```
/*
 * Generated by the Jasper component of Apache Tomcat
 * Version: Apache Tomcat/7.0.23
 * Generated at: 2012-03-21 08:17:38 UTC
 * Note: The last modified time of this file was set to
 *       the last modified time of the source file after
 *       generation to assist with modification tracking.
 */
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class _2_005f6area_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    private static final javax.servlet.jsp.JspFactory _jspxFactory =
        javax.servlet.jsp.JspFactory.getDefaultFactory();    //① 得到 JSP 工厂实现类

    private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

    private javax.el.ExpressionFactory _el_expressionfactory;
    private org.apache.tomcat.InstanceManager _jsp_instancemanager;

    public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
        return _jspx_dependants;
    }

    public void _jspInit() {
        _el_expressionfactory = _jspxFactory.getJspApplicationContext(
            getServletConfig().getServletContext()).getExpressionFactory();
        _jsp_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(
            getServletConfig());
    }

    public void _jspDestroy() {
```

```

    }

    public void _jspService(final javax.servlet.http.HttpServletRequest request,
        final javax.servlet.http.HttpServletResponse response)
        throws java.io.IOException, javax.servlet.ServletException {

        final javax.servlet.jsp.PageContext pageContext;
        javax.servlet.http.HttpSession session = null;
        final javax.servlet.ServletContext application;
        final javax.servlet.ServletConfig config;
        javax.servlet.jsp.JspWriter out = null;
        final java.lang.Object page = this;
        javax.servlet.jsp.JspWriter _jspx_out = null;
        javax.servlet.jsp.PageContext _jspx_page_context = null;

        try {
            response.setContentType("text/html;charset = GB2312");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                                                    null, true, 8192, true); //② 填充 pageContext 变量
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut(); //③ 初始化内置对象
            _jspx_out = out;

            //输出模板文本
            out.write("\r\n");
            out.write("<html>\r\n");
            out.write("<body>\r\n");
            out.write("t"); //

            double r,s;
            String radius = request.getParameter("radius");
            if(radius == null){s = 0.0;}
            else{
                r = Double.parseDouble(radius);
                s = 3.14*r*r;
            } //④ 计算圆面积

            //输出模板文本
            out.write("\r\n");
            out.write("t<font size='5' color='blue'>圆面积计算</font><br>\r\n");
            out.write("t<form action='\" method='post' name='fl'>\r\n");
            out.write("tt 请输入半径 r:t<input type='text\"

```

```
name=\"radius\" value=\\0\" size=\\4\">&nbsp&nbsp&r\n");  
out.write("\\t\\t<input type=\\submit\" value=\\计算\\><br><br>r\n");  
out.write("\\t\\t 面积 S 为: \\t<input type=\\text\" name=\\area\" size=\\7\\>r\n");  
out.write("\\t</form>r\n");  
out.write("</body>r\n");  
out.write("</html>r\n"); //  
out.write("<script language=\\javascript\\>r\n");  
out.write("\\tdocument.fl.area.value=\\");  
out.print(s);  
out.write("\\";r\n");  
out.write("\\tdocument.fl.radius.value=\\");  
out.print(radius);  
out.write("\\";r\n");  
out.write("</script>"); //⑤ 用 javascript 显示计算结果  
} catch (java.lang.Throwable t) {  
    if (!(t instanceof javax.servlet.jsp.SkipPageException)){  
        out = _jspx_out;  
        if (out != null && out.getBufferSize() != 0)  
            try { out.clearBuffer(); } catch (java.io.IOException e) {}  
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);  
    }  
}  
} finally {  
    _jspFactory.releasePageContext(_jspx_page_context);  
}  
}
```

2. JSP 对应的 Servlet 类架构

从以上程序代码可以看出，与 JSP 对应的 Servlet 类继承 `org.apache.jasper.runtime.HttpJspBase` 类，`HttpJspBase` 类由 Tomcat 提供，它实现了 JSP API 中的 `javax.servlet.jsp.HttpJspPage` 接口，该接口继承了 `javax.servlet.jsp.JspPage` 接口，而 `JspPage` 接口又继承了 Servlet API 中的 `javax.servlet.Servlet` 接口。图 2.27 显示了这些类和接口之间的关系。

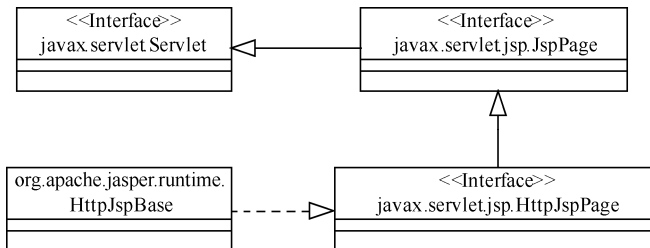


图 2.27 与 JSP 对应的 Servlet 类及接口框图

文件 2_6area.jsp 在运行时首先解析成一个 Java 类 2_005f6area_jsp，该类继承于 org.apache.jasper.runtime.HttpJspBase 基类，而 HttpJspBase 实现了 HttpServlet 接口。

3. JSP 底层机制

分析文件 2 005f6area.jsp.java 的源码不难发现：这其实就是一个 servlet 源文件！它包含

了 Servlet 生命周期各阶段的要素，只是对应 `init()`、`service()`、`destroy()` 3 个方法的名称稍有改动，分别为 `_jspInit()`、`_jspService()`、`_jspDestroy()`，而其实质未变！

进一步分析 `_2_005f6area_jsp.java` 的代码，我们能够大致理清 JSP 的底层运行机制（关键步骤已在源文件中用①②……标出），一共分为五步。

（1）获得 JSP 工厂实现类

调用 `JspFactory` 的 `getDefaultFactory()` 方法获取容器实现（本书中指 Tomcat 7.0）的一个 `JspFactory` 对象的引用。`JspFactory` 是 `javax.servlet.jsp` 包中定义的一个抽象类，其中定义了两个静态方法 `set/getDefaultFactory()`。`set` 方法由 JSP 容器（Tomcat）实例化该页面 Servlet（即 `_2_005f6area_jsp` 类）时置入，所以可以直接调用 `JspFactory.getDefaultFactory()` 方法得到这个 JSP 工厂的实现类。Tomcat 调用 `org.apache.jasper.runtime.JspFactoryImpl` 类。

（2）填充页面上下文（`pageContext`）变量

调用这个 `JspFactoryImpl` 的 `getPageContext()` 方法，填充一个 `PageContext` 后返回，并赋给内置变量 `pageContext`。

（3）初始化内置对象

JSP 页面中内置了几个对象，如 `pageContext`、`session`、`application`、`config`、`out`、`page` 等。可以看出，在 JSP 中的代码片断中直接使用了这些内置对象。事实上，观察 `_jspService()` 方法不难看出，这几个内置对象就是在这里定义的。在对 JSP 文件中的代码片段进行解析之前，先要对这几个内置对象进行初始化。其中 `pageContext` 的内容在第②步中已填充，其他的内置对象都经由该 `pageContext` 得到。具体过程见上面的代码，这里不再赘述。

（4）执行业务逻辑计算

经历了前 3 步，该 JSP 页面的 Servlet 环境就设置完毕了，接下来开始对页面进行解析。`2_6area.jsp` 页面定义了两个 `Double` 类型的变量“`r`”和“`s`”，一个 `String` 类型的变量“`radius`”。用 `request` 的 `getParameter()` 方法得到以字符串形式返回的客户端传来的某个请求参数值，并且使用语句“`Double.parseDouble(radius)`”把这个值转换为 `Double` 类型，然后进行业务逻辑处理（本例为圆面积的计算）。

（5）显示结果

JSP 程序执行完毕后要显示给用户看的不仅是单纯的计算结果（数值），还要在客户端再现整个 XHTML 页面。原 `.jsp` 文件中的 XHTML 文本称为模板文本（`template text`），它会被原封不动地发送到客户端。在前面文件 `_2_005f6area_jsp.java` 的源码中，斜体的代码用于输出模板文本，它们是 Tomcat 容器为 JSP 程序自动生成的，而如果换成 Servlet 程序，这些代码就全都要靠程序员自己手工编写了（就像【例 2.5】那样）。而计算所得的圆面积的数值则使用 JavaScript 脚本显示在页面相应的控件上。

通过这一节的讲解，读者对 JSP 的实现原理和运行机制应该有了比较清楚的了解了，这对 JSP 编程是非常重要的。

2.4 JSP + JavaBean 结构程序

JSP 技术的出现，使得把 Web 应用中的 XHTML 文档和业务逻辑代码有效分离成为可能。通常 JSP 负责动态生成 XHTML 文档，而业务逻辑由其他可重用的组件（如 JavaBean）来实

现。JSP 可通过 Java 程序片段来访问这些业务组件，于是就有了 JSP+JavaBean 的程序结构，这也是最常用的 JSP 开发模式。图 2.28 显示了 JSP 访问服务器端可重用 JavaBean 业务组件的模型。

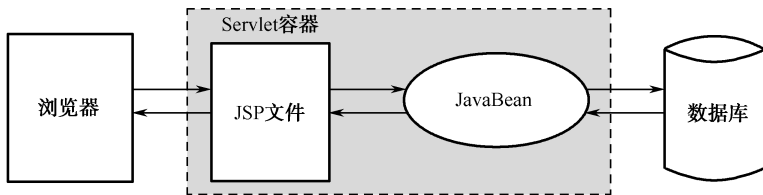


图 2.28 JSP 访问服务器端 JavaBean 组件的模型

【例 2.7】 用 JSP+JavaBean 的方式实现计算圆面积的程序，要求与前例完全相同。

(1) 创建 JavaBean

用记事本编辑名称为 Circle.java 的 JavaBean 文件，该类文件主要实现了圆面积的数学计算操作，具体代码如下：

```

package circle;
public class Circle{
    private double radius=0.0;
    public Circle(){}
    public double getRadius(){
        return radius;
    }
    public void setRadius(double rRadius){
        radius=rRadius;
    }
    public double circleArea(){
        return 3.14*radius*radius;           //实现圆面积的数学计算
    }
}
  
```

在路径 C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ROOT\WEB-INF\classes 下创建名为 circle 的文件夹，将源文件 Circle.java 移到该文件夹下，并编译生成字节码文件 Circle.class。

(2) 创建 JSP 文件

用记事本编辑名称为 2_7area.jsp 的页面文件，该页面文件将实现提示用户输入圆半径及访问 JavaBean 的功能，具体代码如下：

```

<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<body>
<p><jsp:useBean id="circleBean" scope="session" class="circle.Circle"/></p>
<%
    double r,s;
    String radius = request.getParameter("radius");
    if(radius == null){s = 0.0;}
    else{
  
```

```

        r = Double.parseDouble(radius);
        circleBean.setRadius(r);           //向 JavaBean 传递参数
        s = circleBean.circleArea();       //得到 JavaBean 的计算结果
    }
%>
<font size="5" color="blue">圆面积计算</font><br>
<form action="" method="post" name="f1">
    请输入半径 r:  <input type="text" name="radius" value="0" size="4">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
                   <input type="submit" value="计算"><br><br>
    面积 S 为:      <input type="text" name="area" size="7">
</form>
</body>
</html>
<script language="javascript">
    document.f1.area.value="<%=s%>";
    document.f1.radius.value="<%=radius%>";
</script>

```

上段代码中加黑的（形如<jsp:useBean...>）语句是一个 JSP 动作元素，其功能是创建一个“class”属性（这里是 circle.Circle，即第（1）步创建的 JavaBean 文件中的类）所指定的 Bean 类对象，并将该对象命名为“id”属性所指定的值（此处为 circleBean）。

编辑完成后，将这个 JSP 文件保存于 C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ROOT 下。

（3）运行程序

启动 Tomcat，打开 IE 浏览器，输入地址“http://localhost:8080/2_7area.jsp”，结果如图 2.29 所示。



图 2.29 用 JSP+JavaBean 实现圆面积计算

JSP+JavaBean 的开发模式贯彻了 Java 面向对象的编程风格，能够将程序具体的逻辑功能封装于后台的 JavaBean 类中，这在很大程度上实现了页面视图与软件功能模块的相互独立，是当前 Web 领域流行的 MVC 框架开发方式的前身。

2.5 MyEclipse 开发 JSP 程序

2.4 节中【例 2.7】的程序包含两个源文件：JavaBean 文件（Circle.java）和 JSP 文件（2_7area.jsp），随着 Web 应用规模的扩大、功能越来越多，涉及的不同类型的源程序文件也呈指数级增长，如果依然像我们之前那样，采用 Windows 记事本一个个地编辑文件，然后再分别保存到特定目录，其工作量是十分巨大的！而且这种方式也不利于源文件的管理和整个系统的测试。本节介绍如何使用 MyEclipse 工具开发集成的 JSP 程序。

2.5.1 配置 JRE

JRE 是 Java 的运行环境，因此运行 Java 程序时需要 JRE。MyEclipse 中内嵌了 Java 编译器，但这里指定使用 2.1.1 节安装的 JDK7.2 的 JRE，需要进行手动配置。

启动 MyEclipse，选择“Window”→“Preferences...”菜单项，显示 MyEclipse 配置对话框，选择左边目录树中的“Java”→“Installed JREs”，如图 2.30 所示。

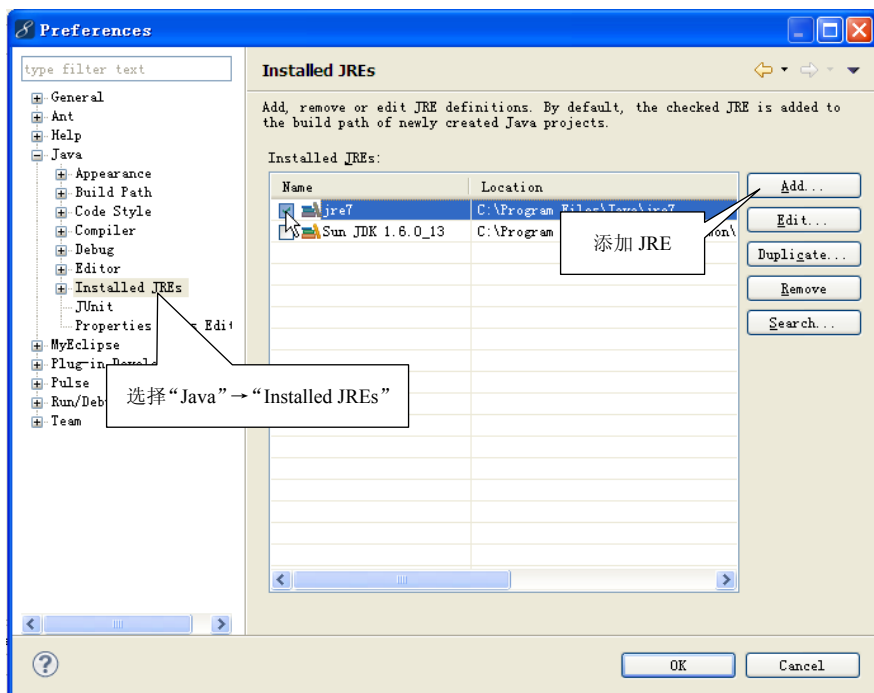


图 2.30 为 MyEclipse 环境配置 JRE

MyEclipse 有默认的 JRE 选项（本书不用），单击右边的“Add...”按钮，添加我们自己安装的 JDK7.2 的 JRE，如图 2.31 所示。

单击“Directory...”按钮，指定 JDK7.2 的 JRE 安装路径“C:\Program Files\Java\jre7”，如图 2.32 所示，单击“确定”按钮。

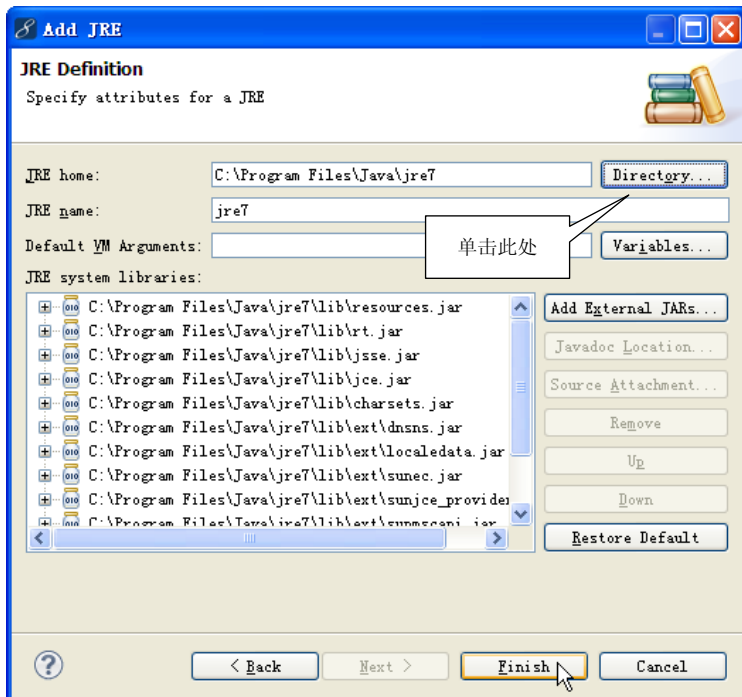


图 2.31 添加 JRE

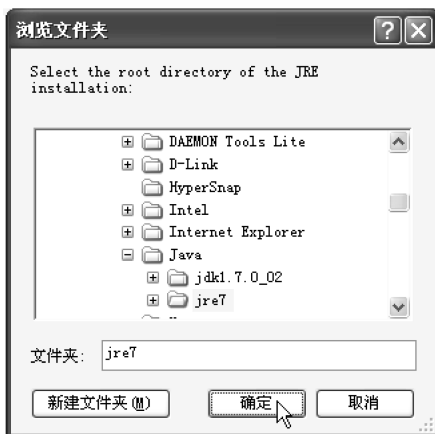


图 2.32 指定 JRE 路径

2.5.2 集成 MyEclipse 与 Tomcat

启动 MyEclipse，选择“Windows”→“Preferences...”菜单项，显示 MyEclipse 配置对话框，选择左边目录树中的“MyEclipse”→“Servers”→“Tomcat”→“Tomcat 7.x”，在右边激活 Tomcat 7.x，设置路径为 2.1.2 节中 Tomcat 7.0 的安装目录，如图 2.33 所示。

继续展开左边目录树中的“MyEclipse”→“Servers”→“Tomcat”→“Tomcat 7.x”→“JDK”，如图 2.34 所示，可见 Tomcat 7.x 默认的运行环境就是 JDK7.2 的 JRE（名为 jre7），如果不是，请读者自行设置（从下拉列表选择即可）。

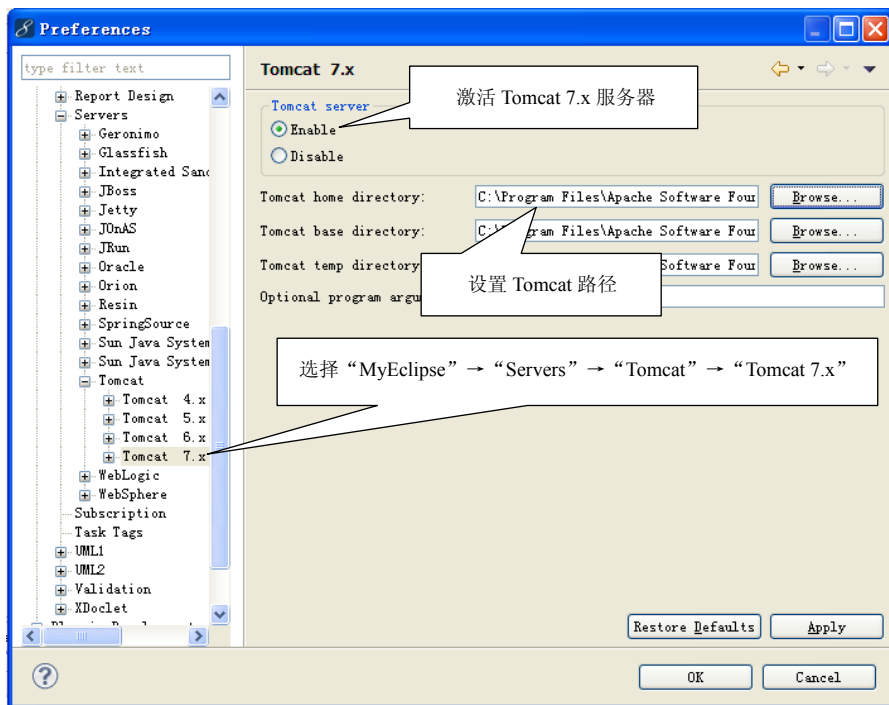


图 2.33 MyEclipse 服务器配置

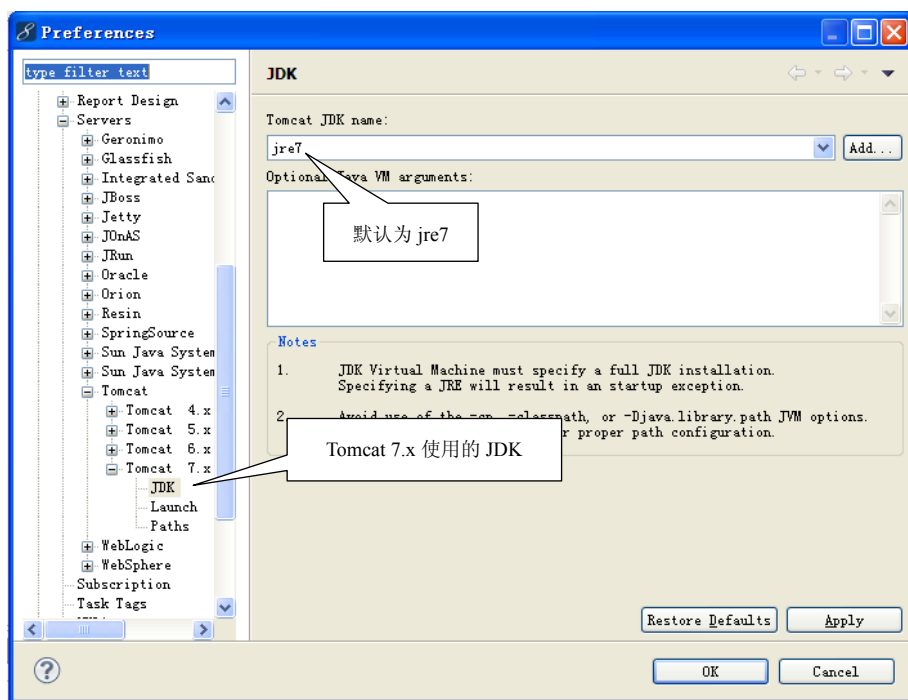


图 2.34 Tomcat 7.x 默认的 JRE 环境

在 MyEclipse 9.0 的工具栏中单击“Run/Stop/Restart MyEclipse Servers”复合按钮，选择“Tomcat 7.x”→“Start”，如图 2.35 所示。

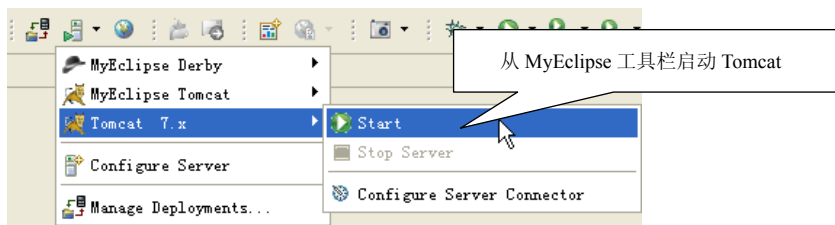


图 2.35 启动 Tomcat 7.x

MyEclipse 主界面下方控制台区会输出 Tomcat 的启动信息，如图 2.36 所示，说明服务器已经开启了。

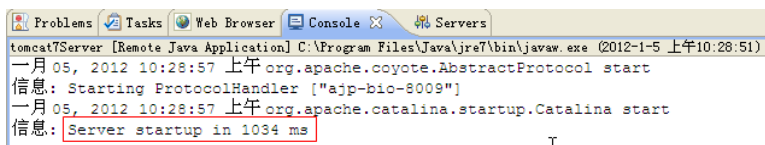


图 2.36 Tomcat 7.0 启动信息

打开浏览器，输入“http://localhost:8080”，如果配置正确，则出现与 2.1.2 节图 2.10 所示一模一样的 Tomcat 7 首页，表示 MyEclipse 9.0 已经与 Tomcat 7.0 紧密集成了！

2.5.3 MyEclipse 开发入门

【例 2.8】用 MyEclipse 工具开发一个与 2.4 节【例 2.7】一模一样的 JSP+JavaBean 结构的程序，功能同样为计算圆面积，要求与前例完全相同。

(1) 创建 Web Project

启动 MyEclipse，选择“File”→“New”→“Web Project”菜单项，为 Web Project 起名，为“2_area”，如图 2.37 所示。

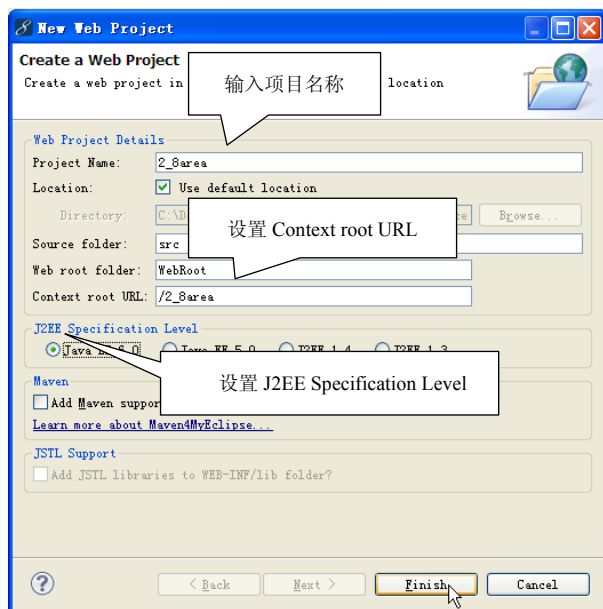


图 2.37 创建 Web Project

除了项目名，MyEclipse 9.0 还要求用户自己指定 Context root URL，这里填写“/2_8area”（即“/”+项目名的形式）。由于本书用的是最新的 Tomcat 7.0 服务器，故 J2EE Specification Level 选项选择最新的 Java EE 6.0。设置完成后单击“Finish”按钮，MyEclipse 会自动生成一个 Web Project 项目。

Web Project，又叫 Web 工程或 Web 项目，所谓 Project 是 MyEclipse 组织管理一个软件程序的诸多源文件的基本集合，本例工程目录树如图 2.38 所示。

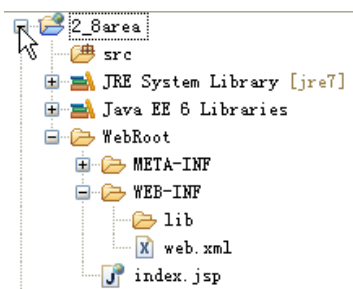


图 2.38 Web 工程目录树

从目录树中可以清楚地看到一个 Web 应用包含什么内容。最简单的 Web 项目只需要一个子目录 WEB-INF，这是一个很重要的目录，Web 项目的配置文件 web.xml 就放在该目录下。通常还会在其下创建 lib 和 classes（目录树中未显示）两个子目录，在 lib 中放置应用依赖的 Java 库文件或自己编写的 jar 包，而 classes 中通常存放编译后的.class 文件。

（2）创建 JavaBean

右击目录树“src”，选择“New”→“Package”菜单项，在 src 目录下创建包 circle，如图 2.39 所示。

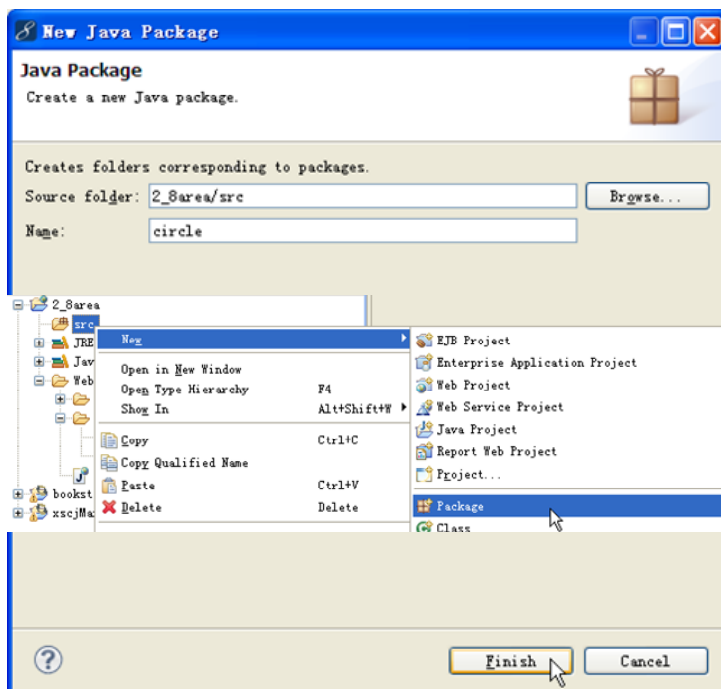


图 2.39 在 src 目录下创建包 circle

右击 circle 包，选择 “New” → “Class” 菜单项，在包中创建类 Circle，如图 2.40 所示。

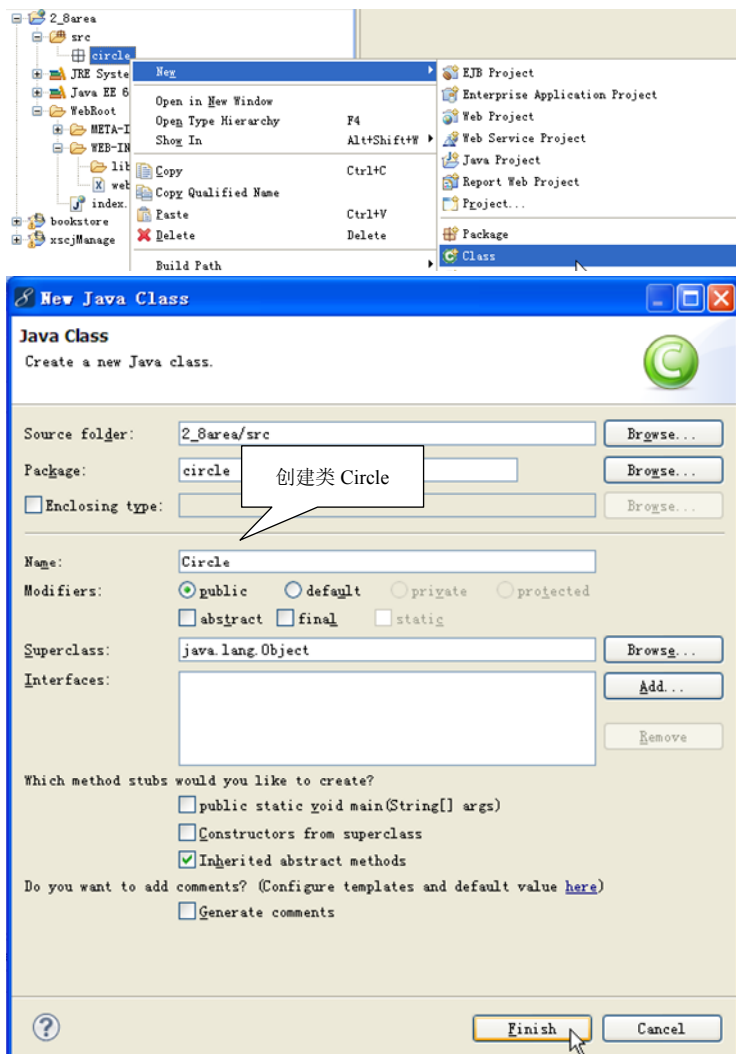


图 2.40 在包 circle 中创建类 Circle

类文件 Circle.java 已经自动生成了类框架，在其中输入代码（见图 2.41 框出部分）。

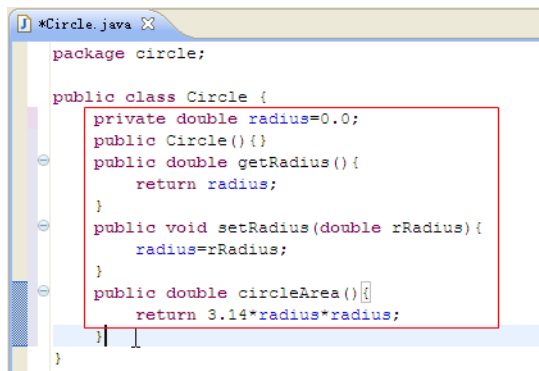


图 2.41 编写 JavaBean 类 Circle 的代码

这个代码与【例 2.7】源文件 Circle.java 的完全相同，都是完成圆面积计算的。

(3) 创建 JSP

右击目录树中的“WebRoot”，选择“New”→“File”菜单项，创建 JSP 文件，取名为 2_area.jsp，如图 2.42 所示。

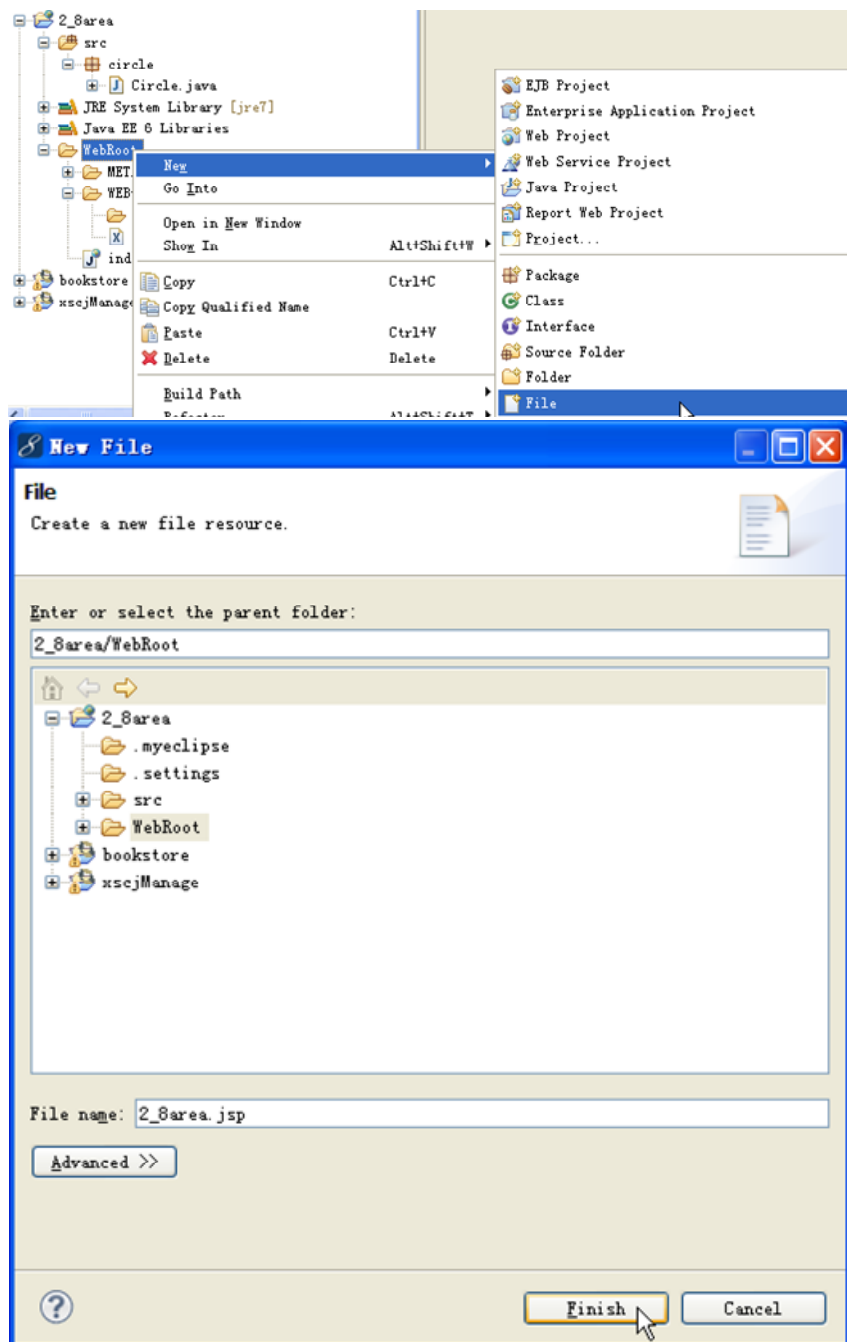


图 2.42 创建 JSP 文件 2_area.jsp

在其中输入代码，代码与【例 2.7】的 2_area.jsp 的完全相同，如图 2.43 所示。

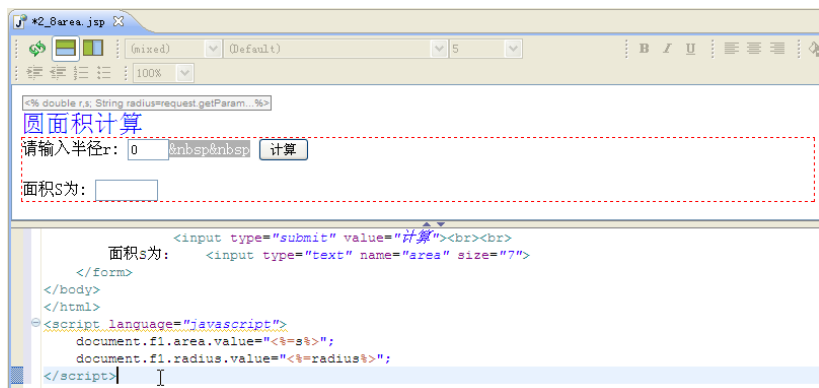


图 2.43 编辑 JSP 代码

(4) 部署

如图 2.44 所示,单击工具栏中的“Deploy MyEclipse J2EE Project to Server...”按钮,将新建的 Web 项目部署到 Tomcat 中。

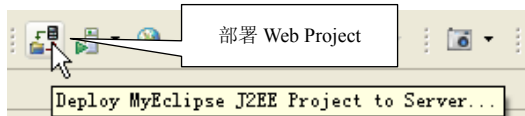


图 2.44 “部署 Web 项目”按钮

从下拉列表中选择项目名为 2_8area 的选项,单击“Add”按钮,选择 Tomcat 7.x 作为服务器,如图 2.45 所示,单击“OK”按钮。

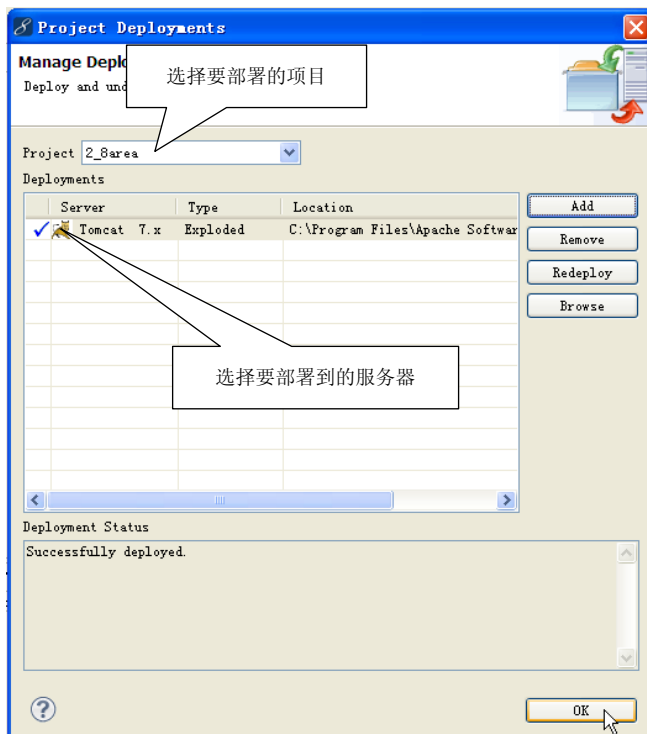


图 2.45 部署项目

(5) 运行浏览

启动 Tomcat 7.x, 在浏览器中输入 `http://localhost:8080/2_area/2_area.jsp`, 将看到如图 2.46 所示的画面。



图 2.46 用 MyEclipse 开发的圆面积计算程序

在本书接下来的章节中, 对于比较简单的例子, 直接用记事本+JDK+Tomcat 方式开发, 而对于较复杂的综合性应用实例, 则会借助于 MyEclipse 集成环境以提高效率。

2.6 上机练习

1. 上网下载 JDK 7 和 Tomcat 7 的最新版本, 按照 2.1 节的指导进行安装与配置。
2. 运行 MyEclipse 9.0 安装文件, 尝试自己破解该软件。
3. 按照 2.5 节的讲述, 搭建 MyEclipse 集成开发环境, 并熟悉这个 IDE 的基本操作。
4. 根据本章实例, 分别用 Servlet、JSP 和 JSP+JavaBean 三种方法开发图 2.47 所示 (与图 2.22 相同) 的圆面积计算程序, 并在实践中体会 JSP 和 JavaBean 的优势。

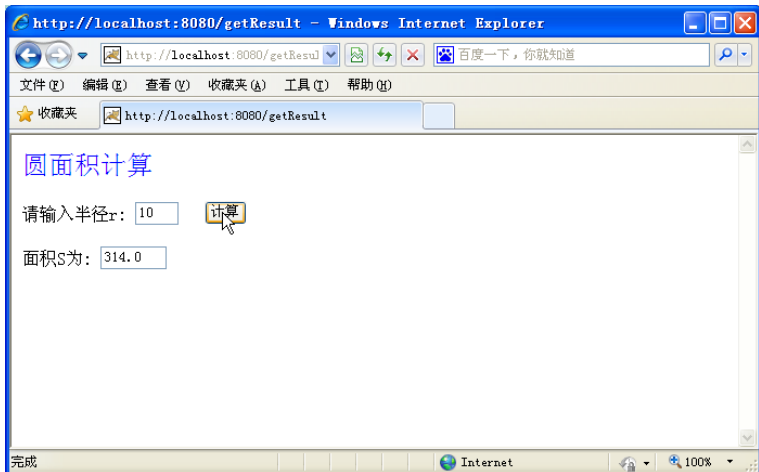


图 2.47 圆面积计算

JSP 编程使用 **Java 语言**编写 tags 和 scriptlets 来封装产生动态网页的处理逻辑。网页能通过 tags 和 scriptlets 访问存在于服务器端的资源。Web 服务器在遇到访问 JSP 页的请求时，首先执行其中的 Java 程序段，这些插入的程序段可以操作数据库、重定向网页等，以实现建立动态网页所需要的功能。

3.1 Java 基础

3.1.1 数据类型、运算符和表达式

1. 数据类型

JSP 用的是 Java 语言，其数据类型与 Java 的一模一样，也包括整型、浮点型、字符型和逻辑型这几大类变量和常量。

(1) 整型

整型变量根据存放数据的范围不同，可分为如表 3.1 所示的几种。

表 3.1 JSP 整型数

类 型	位 数	数值范围（整数）
byte	8	$-2^7 \sim (2^7-1)$
short	16	$-2^{15} \sim (2^{15}-1)$
int	32	$-2^{31} \sim (2^{31}-1)$
long	64	$-2^{63} \sim (2^{63}-1)$

例如：

```
<%  
    byte a=1, b=160;  
    int c=2805;  
%>
```

(2) 浮点型

浮点型分为 float 型和 double 型，如表 3.2 所示。

表 3.2 JSP 浮点数

类 型	位 数	数 值 范 围
float	32	3.4e-038~3.4e+038
double	64	1.7e-308~1.7e+308

表 3.2 中的浮点小数采用了形如 $x.xe\pm xxx$ 的科学计数法来表示。

例如：

```
<%  
    float d=12.8f;  
    double e=12.88E2;  
%>
```

从上面的语句可见，float 型常量后面要跟小写 f，否则会认为是 double 型。

(3) 字符型

字符型即 char 型，在计算机中用十六位无符号数表示，其取值范围为 0~65535，所以可以表示 Unicode 字符集。字符型常量是用单引号引起来的一个字符，转义字符用 “\” 引导。

例如：

```
\ddd    1~3 位八进制数的字符  
\uxxxx  1~4 位十六进制数的字符  
\'       单引号字符  
\       斜杠字符  
\r       回车字符  
\n       换行字符
```

例如：

```
<%  
    char c1='a';  
    char c2=0x61;  
%>
```

(4) 逻辑型

逻辑型（又称布尔型）数据主要用来存放逻辑判断的结果，取值为 true（真）或 false（假）。当将其他数据类型转换为逻辑数据类型时，非 0 转换为 true，0 转换为 false。

例如：

```
<%  
    boolean b1=true;  
    boolean b2=1;  
    boolean b3=1<2;  
%>
```

(5) 常量

常量也可以看成一个变量，其内容固定不变。在 JSP 中定义常量时要加关键字 “final”。

例如：

```
<%  
    final int f1=2085;  
    final char c='a';  
%>
```

【例 3.1】数据类型的演示。

输入以下内容，以 3_1ex.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>数据类型的演示</title>
</head>
<body>
    <%
        float      d=12.8f;
        double     e=12.88E2;
        byte       a=1;
        int        i1=2805;
        int        i2=0xffff;
        char       c1='a';
        char       c2=0x61;
        boolean    b1=true;
        boolean    b2=1<2;
        out.print("d="+ d + "<p></p>");
        out.print("e="+ e + "<p></p>");
        out.print("i1="+ i1 + "<p></p>");
        out.print("i2+a="+ (i2+a) + "<p></p>");
        out.print("c1="+ c1 + "<p></p>");
        out.print("c2="+ c2 + "<p></p>");
        out.print("b1="+ b1 + "<p></p>");
        out.print("b2="+ b2 + "<p></p>");
    %>
</body>
</html>
```

运行结果如图 3.1 所示。

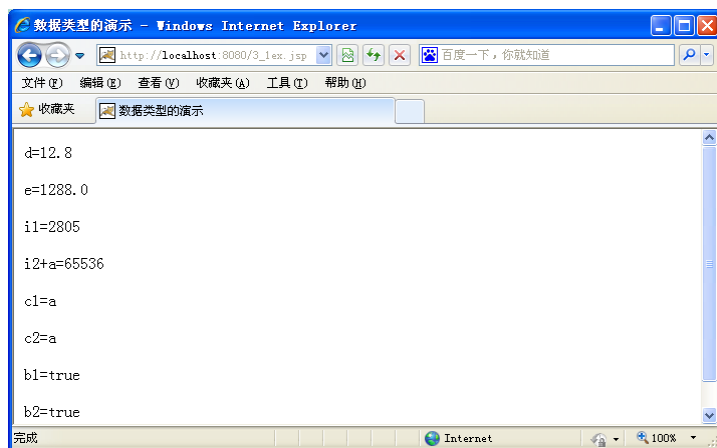


图 3.1 数据类型的演示

其中，out.print()语句在页面上显示括号中表达式的值。

2. 运算符

(1) JSP 中的运算符

JSP 中常用的运算符分为算术运算符、赋值运算符、关系运算符、逻辑运算符、位运算符、条件运算符。

① 算术运算符，如表 3.3 所示。

表 3.3 算术运算符说明

运 算 符	说 明	例 (初值 x=10)	结 果
+	加	x+3	13
-	减	x-3	7
*	乘	x*3	30
/	除	x/3	3
%	取余	x%3	1
++	递增	x++	11
--	递减	x--	9

【例 3.2】算术表达式的演示。

输入以下内容，以 3_2ex.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>算术表达式的演示</title>
</head>
<body>
    <%
        int x=10, y=-1;
        out.print("'x++' + '++y' = "+ (x++ + ++y) + "<p></p>");
        out.print("x/3 + y%3 = "+ (x/3 + y%3) + "<p></p>");
        out.print("(x+y)*3 + y - x/3 = "+ ((x+y)*3 + y - x/3) + "<p></p>");
    %>
</body>
</html>
```

运行结果如图 3.2 所示。

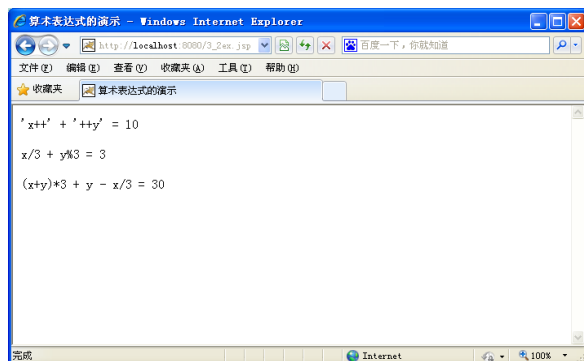


图 3.2 算术表达式的演示

② 赋值运算符，如表 3.4 所示。

表 3.4 赋值运算符说明

运 算 符	说 明	例 (初值 x=10)	结 果
=	等于赋值	x=10	10
+=	加赋值	x+=3	13
-=	减赋值	x-=3	7
=	乘赋值	x=3	30
/=	除赋值	x/=3	3
%=	取余赋值	x%=3	1
.=	字符串赋值	x.="abc"	"10abc"

③ 关系运算符，如表 3.5 所示。

表 3.5 关系运算符说明

运 算 符	说 明	例 (初值 x=10)	结 果
==	等于	x==10	true
!=	不等于	x!=10	false
>	大于	x>12	false
>=	大于等于	x>=8	true
<	小于	x<10	false
<=	小于等于	x<=10	true

④ 逻辑运算符，如表 3.6 所示。

表 3.6 逻辑运算符说明

运 算 符	说 明	例 (初值 x=10)	结 果
&&	与	x==3 && x==10	false
	或	x==3 x==10	true
!	非	! x>3	false

⑤ 位运算符，如表 3.7 所示。

表 3.7 位运算符说明

运 算 符	说 明	例 (二进制)	结果 (二进制)
&	按位与	00101010 & 00010111	00000010
	按位或	00101010 00010111	00111111
~	按位非	~00101010	11010101
^	按位异或	00101010 00010111	00111101
<<	左移	00101010<<2	10101000
>>	右移	00101010>>2 11101010>>2	00001010 11111010
>>>	无符号右移	11101010>>2	00111010

例如：

```
<%
    byte a=7;
    out.println(a>>2);           //显示 “1”
    out.println(a<<2);           //显示 “28”
%>
```

⑥ 条件运算符，语法规则如下：

逻辑表达式 ? 结果 1 : 结果 2

如果“逻辑表达式”为真，则值为结果 1，否则为结果 2。

例如：

```
<%
    int a=2;
    String Msg ;
    Msg=a>2 ? "a 大于 1" : "a 小于等于 2";
    out.println(Msg);           //显示 “a 小于等于 2”
%>
```

(2) 运算符的优先级

对一个包含多种类型的运算符表达式进行计算时，要按运算符的优先顺序从高到低进行，同级的运算符则从左到右进行。运算符的优先顺序如表 3.8 所示。

表 3.8 运算符优先级

优先顺序（依次降低）	运 算 符
一元运算符	++, --, !, ~
乘、除法运算符	*, /, %
加、减法运算符	+, -
移位运算符	>>, >>>, <<
关系运算符	<, >, <=, >=
关系运算符	==, !=
按位与运算符	&
按位异或运算符	^
按位或运算符	
逻辑与运算符	&&
逻辑或运算符	
条件运算符	?:
赋值运算符	=, +=, -=, *=, /=, %=, ^=, &=, =, <<=, >>=, >>>=

在表达式中，可以用括号“()”显式地标明运算顺序，括号中的表达式首先被计算。适当地运用括号可以使表达式的结构更加清晰。

3. 表达式

显示表达式的语法规则如下：

```
<%= expression %>
```

其中, `expression` 是符合 JSP 语法的表达式。在运行后被自动转化为字符串后插入这个表达式的位置显示。因为表达式的值已经被转化为字符串, 所以只能在一行文本中插入这个表达式。

当使用表达式的时候, 应该注意以下几点。

- ① 不能在表达式后面使用分号。
- ② 可以使用任何合法 (即符合 Java 语言规范) 的表达式。
- ③ 如果一个表达式有多个部分, 则计算表达式的值时应该遵循从左到右的规则。

【例 3.3】表达式的应用。

输入以下内容, 以 `3_3ex.jsp` 作为文件名保存:

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>表达式的应用</title>
</head>
<body>
    <%
        String Msg = "这是一个表达式实例";
    %>
    <h4>大家好! </h4>
    <h1><%= Msg %></h1>
</body>
</html>
```

运行结果如图 3.3 所示。

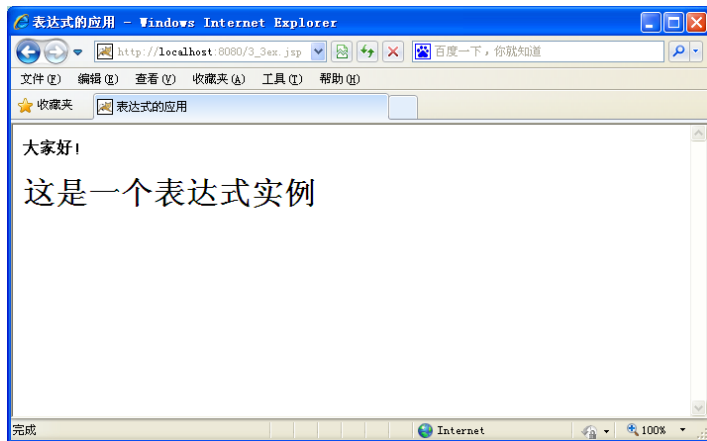


图 3.3 表达式的应用

3.1.2 条件、循环语句

1. 条件语句

条件语句的作用是根据条件表达式决定执行程序的某一部分而不执行另外一部分。`if...else` 语句根据判定条件的逻辑值来确定要执行的语句。当要执行语句包含一条以上时, 需要用 `{}` 括起。

(1) 格式 1

```
if (判断条件) 条件成立执行的语句
```

或

```
if (判断条件)
{
    条件成立执行的语句
}
```

或

```
if (判断条件)
{
    条件成立执行的语句 1
}
else
{
    条件不成立执行的语句 2
}
```

(2) 格式 2

```
if (判断条件 1)
{
    条件 1 成立执行的语句
}
else if (判断条件 2)
{
    条件 2 成立执行的语句
}
else
{
    条件 1 和条件 2 都不成立执行的语句
}
```

注意：else 子句不能单独作为语句使用，它必须和 if 配对，else 总是与在它上面离它最近的 if 或 else if 配对。

【例 3.4】条件语句的应用。

输入以下内容，以 3_4con.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.lang.*,java.util.*" %>
<html>
<head>
    <title>条件语句的应用</title>
</head>
<body>
<center>
    <%
        out.println("<font size=\"4\" color=\"blue\" >今天是:");
        Date date=new Date();
        out.println(date);
    %>
```

//创建日期型对象 date


```
out.println("</font><p></p>");
int day=date.getDay();                                //取得当前系统日期
out.println("<font size='4' color='red' >");
if (day==0) out.println("星期日");
else if (day==1) out.println("星期一");
else if (day==2) out.println("星期二");
else if (day==3) out.println("星期三");
else if (day==4) out.println("星期四");
else if (day==5) out.println("星期五");
else if (day==6) out.println("星期六");
out.println("</font>");

%>
</center>
</body>
</html>
```

运行结果如图 3.4 所示。

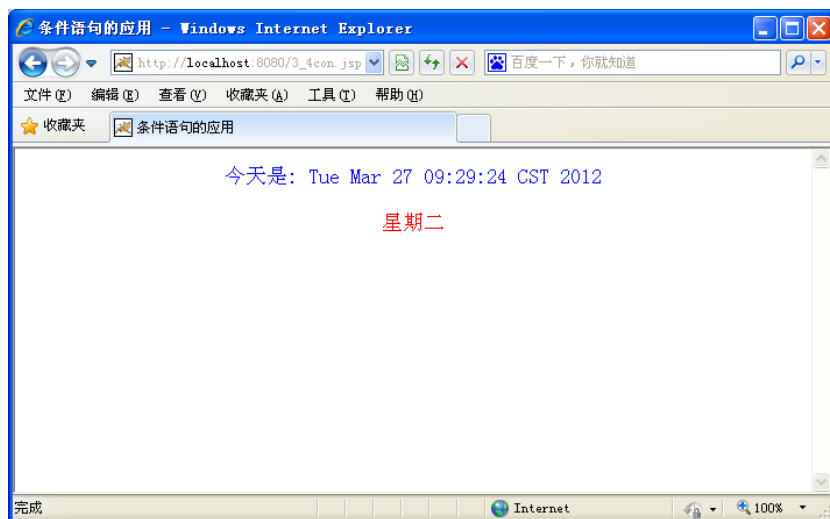


图 3.4 条件语句的应用

2. 循环语句

循环语句的作用是反复执行一段代码，直到满足终止循环的条件为止。一个循环一般应包括两部分内容。

- 循环体：这是反复循环的一段代码，可以是单一的一条语句，也可以是复合语句。
- 循环终止条件：通常是一个条件表达式，以确定循环是否终止。

JSP 循环主要有 for 语句、while 语句和 do...while 语句三种形式。

(1) for 语句

格式：

```
for(初始化; 循环终止条件; 迭代)
{
    循环体
}
```

这里的初始化在第一次循环开始前执行,用来设置循环的一些初始条件。迭代部分是在当前循环结束、下一次循环开始前执行的语句,常用来使计数器加 1 或减 1。

“初始化”、“循环终止条件”和“迭代”这三部分都可以是空语句,但是分号不能省略。终止条件为空的时候,相当于一个无限循环。

【例 3.5】 for 循环语句的应用。

输入以下内容,以 3_5rep1.jsp 作为文件名保存:

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.lang.*" %>
<html>
<head>
    <title>for 循环语句的应用</title>
</head>
<body>
    <%
        out.print("<font size=\"4\" color=\"green\">");
        out.println("<h3>10-100 之间不能被 3 整除的数</h3>");
        int n, js=0;
        for (n=10; n<=100; n++)
        {
            if (n%3==0)    continue;
            js=js+1;
            out.println( n+",");
        }
        out.print("</font >");
    %>
</body>
</html>
```

运行结果如图 3.5 所示。

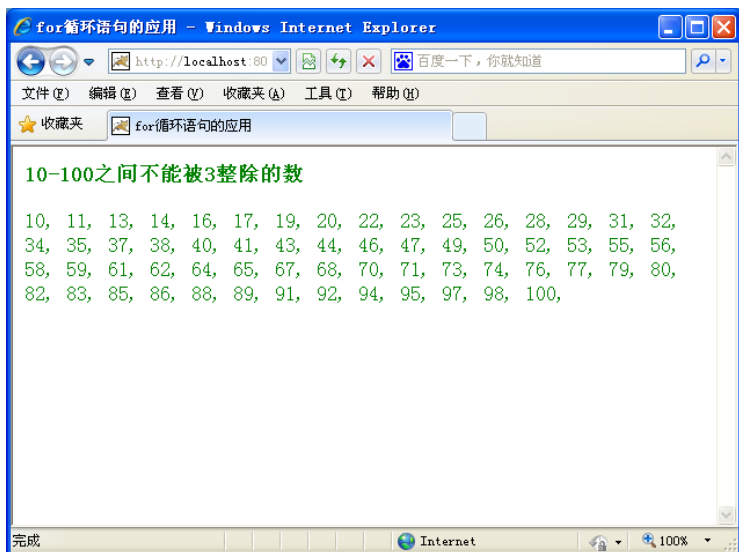


图 3.5 for 循环语句的应用

(2) while 语句

格式:

```
while (循环条件)
{
    循环体
}
```

循环条件成立时（逻辑表达式为 true），循环体被执行。while 每次循环前都要判断“循环条件”，如果“循环条件”的值为 false，循环结束。

(3) do...while 语句

格式:

```
do{
    循环体
}
while (循环终止条件);
```

do...while 语句首先执行循环体中的语句，然后再判断“循环终止条件”。也就是说，循环体至少被执行一次。

【例 3.6】 for、while、do...while 循环语句的应用。

输入以下内容，以 3_6rep2.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.lang.*" %>
<html>
<head>
    <title>循环语句的应用</title>
</head>
<body>
    <%
        out.print("<font size='4' color='green'>");
        out.println("for 循环语句<p></p>");
        int i=1,n=1;
        for (i=1; i<=10; i++)
        {
            n*=i;
        }
        out.println("10!="+n+"<p></p>");
        out.println("While 循环语句<p></p>");
        i=1; n=1;
        while (i<=10)
        {
            n*=i;
            i++;
        }
        out.println("10!="+n+"<p></p>");
        out.println("do...While 循环语句<p></p>");
        i=1; n=1;
        do{

```

```

        n*=i;
        i++;
    }while (i<=10);
    out.println("10!="+n+"<p></p>");
    out.print("</font >");

%>
</body>
</html>

```

运行结果如图 3.6 所示。

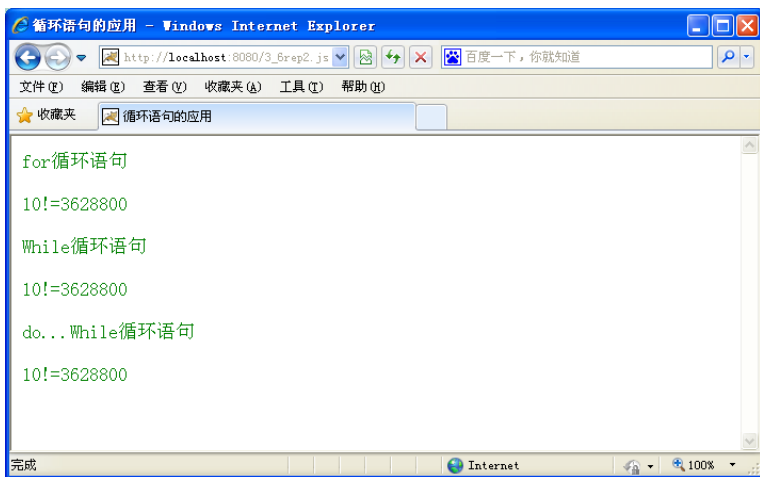


图 3.6 循环语句的应用

在循环体部分，都可以用 `break`、`continue` 语句控制循环的流程。其中 `break` 用于强行退出循环；而 `continue` 则跳到循环结束位置，不执行循环体中剩余的语句。

① `break` 语句。

格式：

```
break [label];
```

如果没有 `label` 的标号，则 `break` 使得程序从循环体中跳出来；如果有 `label` 标号，则程序跳转到标号所指明的位置。

② `continue` 语句。

格式：

```
continue [label];
```

如果没有 `label` 的标号，`continue` 语句结束本次循环，跳过循环体，接着进行循环终止条件的判断；如果有 `label` 标号，`continue` 则可以跳转到标号所指明的位置。

例如，以下代码显示 1~10 的奇数：

```

<%
    for (int i=1; i<=10; i++)
    {
        if (i%2==0) continue;
        out.print(i+" ");
    }
%>

```

3.1.3 自定义函数、变量声明

1. 自定义函数

格式:

```
<%!  
    返回类型 函数名(类型 参数, ...)  
    {  
        语句  
        return(返回值);  
    }  
%>
```

自定义函数可以传送参数，其返回值通过 `return` 语句得到，调用方法与系统函数相同。

例如:

```
<%!  
    int sum (int a)  
    {  
        int b=a+10;  
        return (b);  
    }  
%>  
<%  
    out.println(sum(2));  
%>
```

2. 全局变量和局部变量的声明

`<%! ... %>`和主程序中所声明的变量为全局变量，作用于整个页面文件。自定义函数中声明的变量为局部变量，只能在自定义函数内部应用。

例如:

```
<%!  
    int a=1,b=2,c=3;    //全局变量  
    int sum ()  
    {  
        int d=a+b+c;    //局部变量  
        return (d);  
    }  
%>  
<%  
    int e=6;            //全局变量  
%>
```

其中，`<%!... %>`声明的变量 `a`, `b`, `c` 和程序声明区的变量 `e` 是全局变量，函数中声明的变量 `d` 为局部变量。

3.1.4 数组

数组是最简单的数据集合形式，在一个数组中，多个变量使用一个变量名和各自的编号命名。JSP 使用字符 ‘[’ 和 ‘]’ 标记数组。

声明格式：

数组类型 数组名[]=new 数组类型[元素数量]

例如：

```
<%
    int  a[ ] = new int[3];    //声明一个 3 个元素的整型数组 a[0], a[1], a[2]
    float b[ ] = new float[4]; //声明一个 4 个元素的浮点数数组 b[0], b[1], b[2], b[3]
    char c[ ] = new char[3];  //声明一个 3 个元素的字符数组 c[0], c[1], c[2]
    a[1] = 1;
    b[2]=2.8;
    char c[ ]={ 'a', 'b', 'c' };
%>
```

JSP 可以使用多维数组，多维数组的声明与赋值方法如下：

```
<%
    int a[ ][ ] = new int[2][15];
    int [ ][ ] b = new int[3][4];
    int c[ ][ ] = {{11,12}, {21,22}, {31,32}};
%>
```

【例 3.7】冒泡排序法。

输入以下内容，以 3_7array.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<html>
<head>
    <title>数组的应用--冒泡排序法</title>
</head>
<body>
<hr>
    <%
        int i,j,temp;
        int intArray[ ] = {50, -6, 20, 1, 15 };
        int len= intArray.length;
        out.println("排序前的数据是: <br>");
        for(i=0; i<len; i++)
        {
            out.println(intArray[i]+ " ");
        }
        for (i=0; i<len-1; i++)
        for (j=i+1; j<len; j++)
        if(intArray[i]>intArray[j])
        {
            temp = intArray[i];
```

```
        intArray[i] = intArray[j];
        intArray[j] = temp;
    }
    out.println("<br>排序后的数据是: <br>");
    for(i=0; i<len; i++)
    {
        out.println(intArray[i]+ "");
    }
%>
</body>
</html>
```

运行结果如图 3.7 所示。

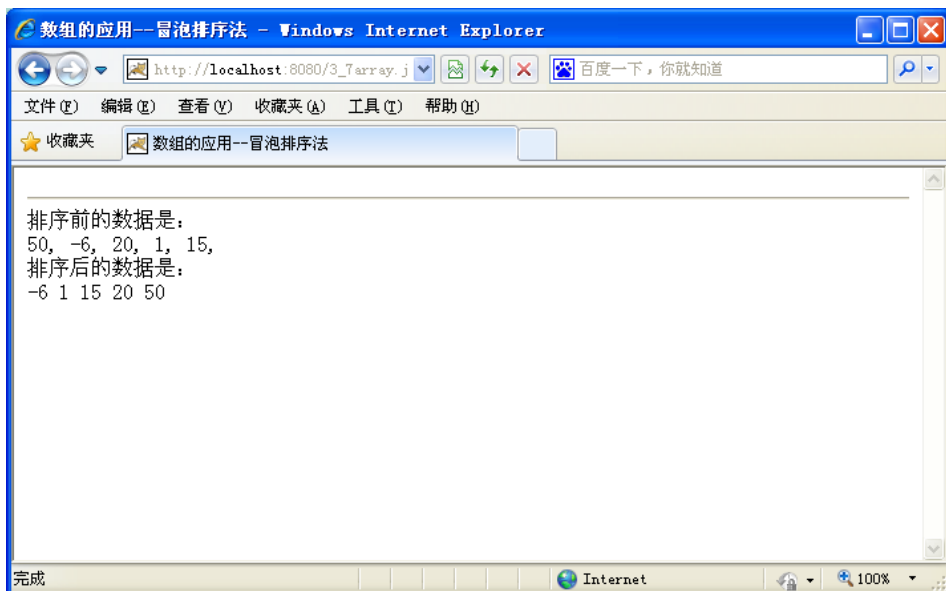


图 3.7 数组的应用--冒泡排序法

本例用到了一维数组的赋值语句: `intArray[] = {50, -6, 20, 1, 15 }`, 数组元素的比较语句: `intArray[i]> intArray[j]`, 数组长度的获取语句: `int len= intArray.length`。可见, 数组元素完全可以当做一个变量来使用。

3.1.5 面向对象程序设计

1. 类和对象

类是一个创建对象的模板, 包含属性和方法。JSP 的类包括系统已经定义好的类和用户自定义类。类实例化后就是对象。

例如:

```
<%
    java.util.Date date=new java.util.Date();
    out.println(date);
%>
```

Date 是一个系统已经定义好的类，date 是 Date 类生成的一个对象。上例代码用来显示系统日期和时间。JSP 与 Java 语言公用同一套类库系统。有关 JSP 系统常用类将在 3.2 节详细介绍。用户还可以自定义类，自定义类的语法如下：

```
class 类名
{
    定义属性;
    ...
    定义方法;
    ...
}
```

例如，自定义一个手机类 phone：

```
class phone
{
    String name;
    String price;
    String phone_name (String phone_name)
    {
        this.name = phone_name;
        return(this.name);        //this 代表对象本身
    }
    String phone_price(String phone_price)
    {
        this.price= phone_price;
        return(this.price);
    }
}
```

此类中定义了手机的两个属性（“name”和“price”）和两个方法（“phone_name”和“phone_price”）：

对类进行实例化，语法如下：

```
类名 对象名 = new 类名();
```

例如：

```
phone Myphone = new phone();
```

实例化后就可以使用该类的方法了，格式如下：

```
对象名. 类方法(参数);
```

例如：

```
out.print(Myphone.phone_name("Amoi")+"<p>");    //设置手机品牌
out.print(Myphone.phone_price("1200 元")+"<p>");    //设置手机价格
```

复制对象，格式如下：

```
类名 新对象名 = 要复制的对象名;
```

例如：

```
phone Yourphone = Myphone;
```

2. 继承与构造函数

(1) 继承

继承是面向对象程序设计的一个重要概念。在已有的一个类的基础上，建立一个与其相关的新类，这种关系称为继承。原有的类为父类，新建的类则为子类。

继承的语法如下：

```
class 子类名 extends 父类名
{
    定义属性;
    ...
    定义方法;
    ...
}
```

【例 3.8】继承的应用。

输入以下内容，以 3_8inher.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<html>
<head>
    <title>继承的应用</title>
</head>
<body>
    <%
        class phone
        {
            String name;
            String price;
            String phone_name(String phone_name)
            {
                this.name = phone_name;
                return(this.name);
            }
            String phone_price(String phone_price)
            {
                this.price = phone_price;
                return(this.price);
            }
        }
        class user extends phone
        {
            String user;
            String phone_user(String user)
            {
                this.user=user;
                return(this.user);
            }
        }
    %>
    <%
        user zaq=new user();
        out.print("姓名： ");
        out.print(zaq. phone_user("zaq")+"<p>");
```

```
out.print("手机品牌: ");
out.print(zaq.phone_name("iPhone 4S"))+"<p>");
out.print("价格: ");
out.print(zaq.phone_price("6200 元"))+"<p>");

%>
</body>
</html>
```

运行结果如图 3.8 所示。

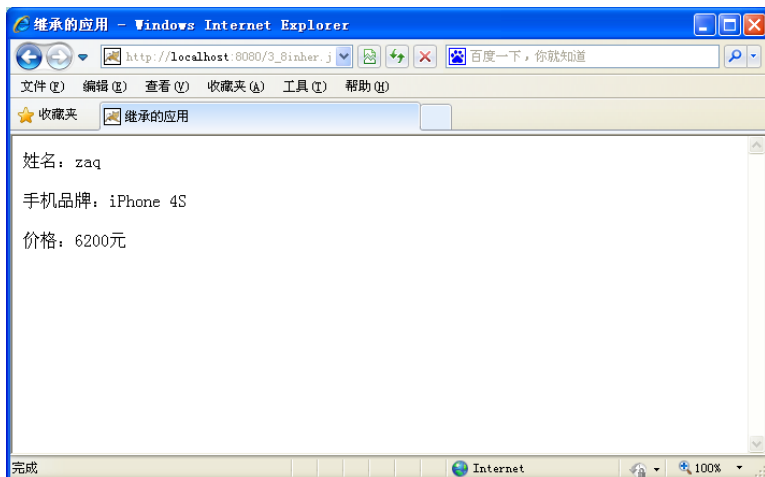


图 3.8 继承的应用

(2) 构造函数

构造函数是定义在类中的同名方法，在生成新对象时会自动执行该方法。

【例 3.9】构造函数的应用。

输入以下内容，以 3_9struct.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>构造函数的应用</title>
</head>
<body>
    <%
        class phone
        {
            String name;
            String price;
            phone()                //构造函数，对属性进行初始化
            {
                name="iPhone 4S";
                price="6200 元";
            }
            String phone_name(String phone_name)
            {
```

```
        this.name = phone_name;
        return(this.name);
    }
    String phone_price(String phone_price)
    {
        this.price = phone_price;
        return(this.price);
    }
}
phone my = new phone();           //生成对象 my 时已对.name 和.price 属性赋值了
out.print("手机品牌: ");
out.print(my.name+"<p>");
out.print("价格: ");
out.print(my.price+"<p>");

%>
</body>
</html>
```

运行结果如图 3.9 所示。

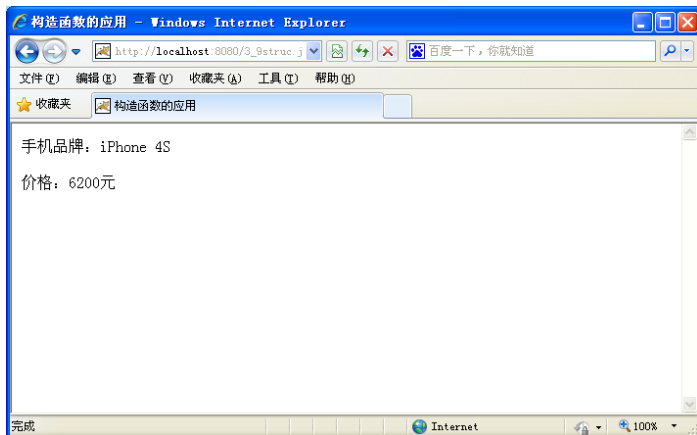


图 3.9 构造函数的应用

学完本节，大家肯定已经发现：JSP 所用的 Java 代码就是普通的 Java 代码，只不过程序运行结果换以 Web 网页的方式显示而已。因此 JSP 具备了 Java 语言的简单易用、完全面向对象、平台无关、安全可靠、主要面向互联网的特点。自 JSP 推出后，众多大公司都支持 JSP 技术的服务器，如 IBM、Oracle、Bea 等，所以 JSP 迅速成为商业应用的服务器端语言。

3.2 JSP 系统常用类

3.2.1 常用数值类

JSP 的常用数值类有 Integer 类、Float 类、Math 类和 Random 类等，它们大部分属于 java.lang 包。

1. Integer 类

`Integer` 类的方法常用于整型与字符串的相互转化、整型数与进位法的转换等。方法说明如表 3.9 所示。

表 3.9 `Integer` 类方法说明

方 法	说 明
<code>compareTo(int)</code>	比较两数大小。前者比后者大为 1，小为-1，相等为 0
<code>parseInt(String)</code>	转换成整数
<code>decode(String)</code>	转换字符串为整数
<code>equals(Object)</code>	比较两数是否相等
<code>toBinaryString(int)</code>	转换成二进制数字字符串
<code>toOctalString(int)</code>	转换成八进制数字字符串
<code>toHexString(int)</code>	转换成十六进制数字字符串
<code>floatValue()</code>	返回浮点数值
<code>intValue()</code>	返回整数数值
<code>valueOf(String)</code>	转换字符串为整型

【例 3.10】`Integer` 类方法的应用。

输入以下内容，以 `3_10int.jsp` 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>integer 类的应用</title>
</head>
<body>
    <%
        int a=200, b=100;
        String s=" ";
        Integer int1=new Integer(a);
        Integer int2=new Integer(b);
        out.print("整数对象 int1: "+int1+"<br>");
        out.print("字符串对象 int2: "+int2+"<br>");
        out.print("<hr>");
        int ic=int1.compareTo(int2);
        if (ic==1){
            s="a>b";
        } else if (ic==0){
            s="a=b";
        } else {
            s="a<b";
        }
        out.print("int1 和 int2 比较结果: "+s+"<br>");
        out.print("int1 转化为浮点数值: "+int1.floatValue()+"<br>");
        out.print("将整数 int1 转为 16 进制字符串: "+Integer.toHexString(int1)+"<br>");
    %>
    </body>
</html>
```

```
%>
</body>
</html>
```

运行结果如图 3.10 所示。



图 3.10 Integer 类应用示例

2. Float 类

Float 类的方法常用在字符串与浮点数的相互转化、判断相同、转化为整型数等。方法说明如表 3.10 所示。

表 3.10 Float 类方法说明

方 法	说 明
compareTo(float)	比较两数大小
equals(Object)	比较两数是否相等
toString()	转换成字符串
floatValue()	返回浮点数值
intValue()	返回整数数值
valueOf(String)	将字符串转换成 Float

【例 3.11】Float 类方法的应用。

输入以下内容，以 3_11float.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<html>
<head>
  <title>float 类的应用</title>
</head>
<body>
  <%
    float a=10.5f, b=-12.5f;
    String s="15.22";
    Float f1=new Float(a);
    Float f2=new Float(b);
```

```

String str=new String();
out.print("浮点数对象 f1: "+f1+"<br>");
out.print("浮点数对象 f2: "+f2+"<br>");
out.print("<hr>");
if (f1.compareTo(f2)==1){
    str="f1>f2!";
}else if (f1.compareTo(f2)==-1){
    str="f1<f2!";
}else{
    str="f1=f2!";
}
out.print("f1 和 f2 比较的结果" + str + "<br>");
out.print("f1 浮点数对象的数值: "+f1.floatValue()+"<br>");

%>
</body>
</html>

```

运行结果如图 3.11 所示。

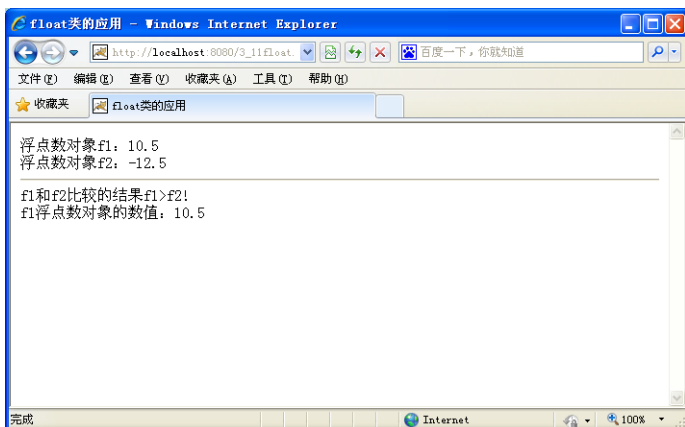


图 3.11 Float 类的应用

3. Math 类

Math 类提供了常用的数学方法，如四舍五入、取绝对值、弧度角度转换等。方法说明如表 3.11 所示。

表 3.11 Math 类方法说明

方 法	说 明
round(double)	返回四舍五入后的整数
abs(long)	返回绝对值
max(Object, Object)	返回最大值
min(Object, Object)	返回最小值
sqrt(double)	返回平方根
log(double)	返回自然对数
pow(double1, double2)	返回 double1 的 double2 次方

续表

方 法	说 明
toDegrees(double)	返回角度
toRadians(double)	返回弧度
sin(double)	返回正弦值
cos(double)	返回余弦值
tan(double)	返回正切值
random()	返回随机数

【例 3.12】 Math 类方法的应用。

输入以下内容，以 3_12math.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>math 类的应用</title>
</head>
<body>
    <%
        double pi=Math.PI;
        int ran;
        out.print("数值-11.2 的绝对值为: "+Math.abs(-11.2)+"<br>");
        out.print("取 11.2 和 22.1 两数的最大值为: "+Math.max(11.2, 22.1)+"<br>");
        ran=(int)(Math.random()*10);
        out.print("产生一个介于 1 至 10 之间的随机整数: "+ran+"<br>");
    %>
</body>
</html>
```

运行结果如图 3.12 所示。

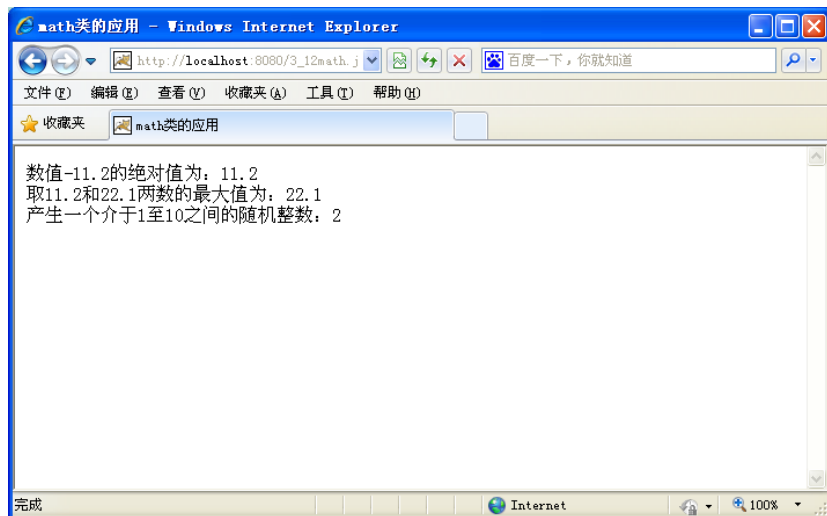


图 3.12 Math 类的应用

4. Random 类

Random 类方法产生一个 Random 对象。使用它所提供的方法，可以产生随机整数、随机浮点数、随机双精度数、随机长整数。方法说明如表 3.12 所示。

表 3.12 Random 类方法说明

方 法	说 明
nextInt()	返回随机整数
nextFloat()	返回随机浮点数
nextDouble()	返回随机双精度数
nextLong()	返回随机长整数

【例 3.13】 Random 类方法的应用。

输入以下内容，以 3_13random.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.util.*" %>
<html>
<head>
    <title>random 类的应用</title>
</head>
<body>
    <%
        Random ran= new Random();
        out.print("产生随机整数为： <br>");
        out.print(ran.nextInt());
    %>
</body>
</html>
```

运行结果如图 3.13 所示。

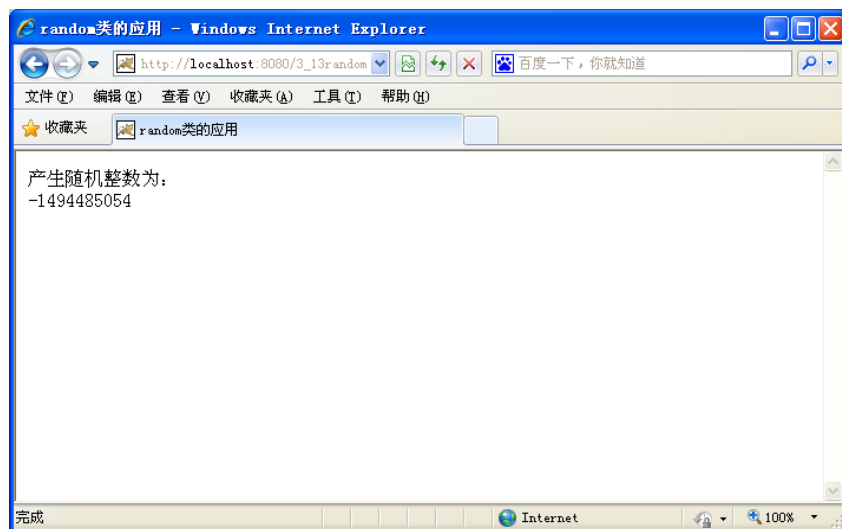


图 3.13 Random 类的应用

3.2.2 常用字符串类

1. Character 类

Character 类用于提供单个字母的处理、判断与转换的方法，属于 java.lang 包。方法说明如表 3.13 所示。

表 3.13 Character 类方法说明

方 法	说 明
charValue()	返回对象的字符
compareTo(Object)	返回本对象与指定对象的 ASCII 差值，返回值为整数类型
equals(Object)	返回本对象与指定对象是否相同，返回 true 或 false
getType(char)	返回变量的字符种类并以整数表示，1：大写，2：小写，3：数字
isDigit(char)	判断变量是否为十进制数
isLowerCase(char)	判断字符是否为小写字母
isUpperCase(char)	判断字符是否为大写字母
toLowerCase(char)	将字符转换为小写
toUpperCase(char)	将字符转换为大写

【例 3.14】Character 类方法的应用。

输入以下内容，以 3_14char.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>character 类应用</title>
</head>
<body>
    <%
        char c1='A';
        char c2='b';
        Character oc1=new Character(c1);
        Character oc2=new Character(c2);
        out.print("定义字符对象 c1 为： " + oc1.charValue() + "<br>");
        out.print("定义字符对象 c2 为： " + oc2.charValue() + "<br>");
        if (Character.getType(c1)==1){
            out.print("c1 是大写！ <br>");
        }
        if (oc2.equals(c1)){
            out.print("c1 字符与 c2 字符相同！ <br>");
        } else {
            out.print("c1 字符与 c2 字符不相同！ <br>");
        }
    %>
</body>
</html>
```

运行结果如图 3.14 所示。

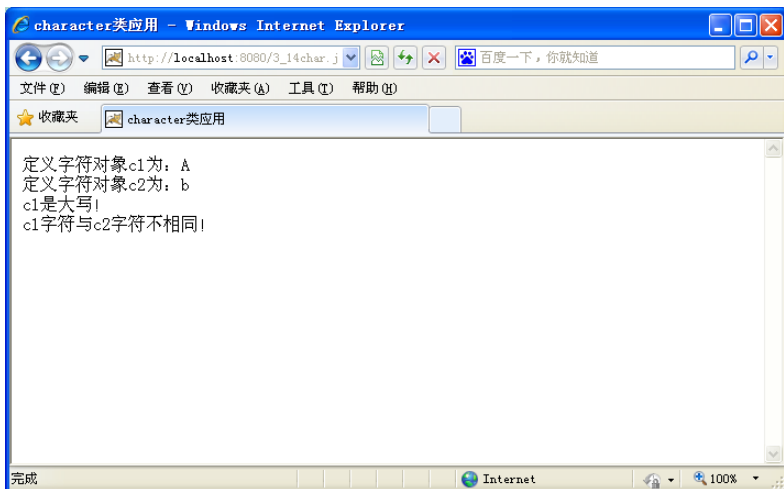


图 3.14 Character 类应用

2. String 类

String 类用于字符串处理，可以进行“字符串比较”、“字符转换”、“字符串搜索”和“字符串连接及插入”。

(1) 基本方法

String 类的基本方法如表 3.14 所示。

表 3.14 String 类基本方法说明

方 法	说 明
length()	返回字符串对象的字符长度，为整数类型
charAt(int)	返回字符串对象于指定位置上的字符
getChars(int_start, int_end, char[], int)	从当前字符串中复制若干字符到 char[] 中

【例 3.15】String 类方法的应用。

输入以下内容，以 3_15string.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>string 类应用</title>
</head>
<body>
    <%
        String str;
        str=new String("http://www.njnu.edu.cn/MCS");
        char c[]=new char[22];
        out.print(str+" 字符串对象长度为: "+str.length()+"<br>");
        out.print("第"+25+"个字符为: "+str.charAt(25)+"<br>");
        out.print("<hr>");
        //复制字符串字符到字符数组 c 中
```

```

        str.getChars(0, 22, c, 0 );
        //输出字符数组 c 中的字符
        out.print("c 数组的值为: " + String.valueOf(c).trim() + "<br>");
        out.print("<hr>");
    %>
</body>
</html>

```

运行结果如图 3.15 所示。

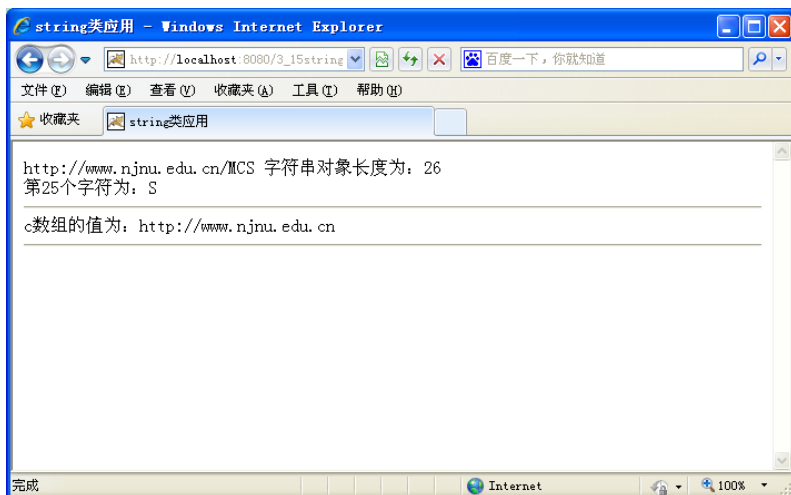


图 3.15 String 类应用

(2) 比较处理方法

String 类比较处理方法如表 3.15 所示。

表 3.15 String 类比较处理方法说明

方 法	说 明
compareTo(String)	比较两个字符串大小
startsWith(String)	判断开始位置的字符串是否为指定字符串
endsWith(String)	判断结束字符串是否为指定字符串
equals(Object)	判断两个字符串是否相同
regionMatches(int_start,String ,int_start,int_len)	判断两个字符串子串是否相等

【例 3.16】字符串比较处理方法的应用。

输入以下内容，以 3_16strcom.jsp 作为文件名保存：

```

<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>字符串比较处理方法的应用</title>
</head>
<body>
    <%
        String str1, str2 ;

```

```

str1=new String("Hello");
str2=new String("Hello World");
out.print("str1 字符串对象为: " + str1 + "<br>");
out.print("str2 字符串对象为: " + str2 + "<br>");
out.print("<hr>");
if(str1.equals(str2)){
    out.print("str1 字符串与 str2 字符串相同<br>");
} else {
    out.print("str1 字符串与 str2 字符串不相同<br>");
}
if(str2.regionMatches(0, str1, 0, str1.length())){
    out.print("str2 字符串包含 str1<br>");
} else {
    out.print("str2 字符串不包含 str1<br>");
}
%>
</body>
</html>

```

运行结果如图 3.16 所示。

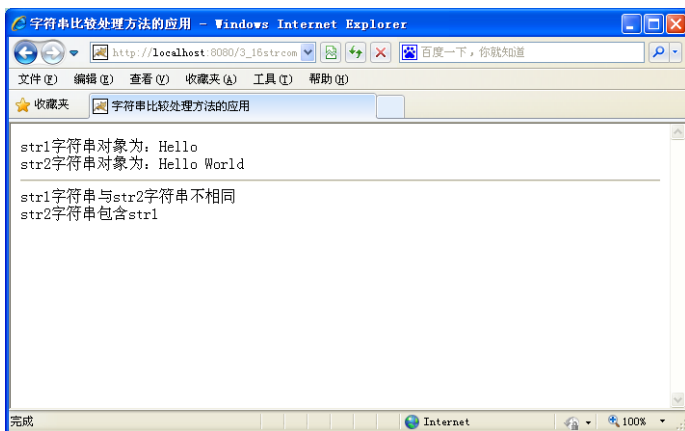


图 3.16 字符串比较处理方法的应用

(3) 搜索方法

字符转换及字符串搜索方法如表 3.16 所示。

表 3.16 字符转换及字符串搜索方法说明

方 法	说 明
replace(char, char)	替换字符串中的指定字符
toLowerCase()	将字符串对象中的字母都转换为小写字母
toUpperCase()	将字符串对象中的字母都转换为大写字母
trim()	去除字符串前后的空格符
toString()	返回字符串对象
indexOf(String)	在字符串对象中搜索第一次出现指定字符串的位置

续表

方 法	说 明
lastIndexOf(String)	在字符串对象中搜索最后一次出现指定字符串的位置
substring(int_start,int_end)	选取字符串对象指定起始位置到结束位置的内容
concat(String)	合并字符串

【例 3.17】 字符转换及字符串搜索方法的应用。

输入以下内容，以 3_17strtran.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>字符转换及字符串搜索方法的应用</title>
</head>
<body>
    <%
        String str1,str2;
        str1=new String("String class");
        str2=new String("I Love JSP!");
        out.print("str1 字符串对象: "+str1+"<br>");
        out.print("str2 字符串对象: "+str2+"<br>");
        out.print("<hr>");
        out.print("更换 str1 字符串对象的 S 字符为 A: "+str1.replace('S','A')+"<br>");
        out.print("更换 str2 字符串对象为小写: "+str2.toLowerCase()+"<br>");
        if(str2.indexOf("JSP")!= -1)
            out.print("str2 字符串对象中搜索 JSP 的位置:"+str2.indexOf("JSP")+"<br>");
        else
            out.print("在 str2 字符串对象中搜索不到该字符串! "+<br>");
        out.print("截取 str2 字符串对象第 2 到 10 位置的字符: "+str2.substring(2,11)+"<br>");
    %>
</body>
</html>
```

运行结果如图 3.17 所示。

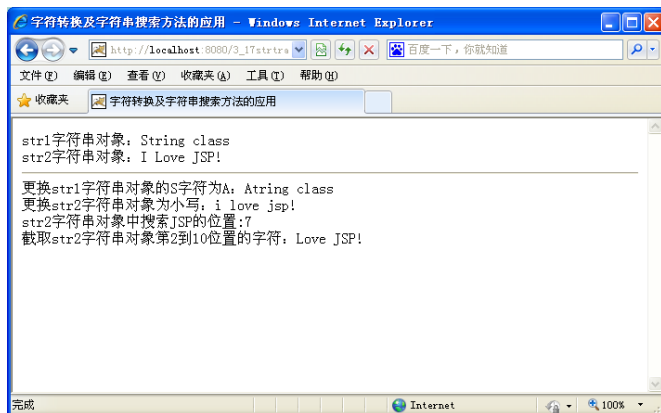


图 3.17 字符转换及字符串搜索方法的应用

3. StringBuffer 类

StringBuffer 类多重连接及插入方法如表 3.17 所示。

表 3.17 字符串多重连接及插入方法说明

方 法	说 明
append(String)	将准备插入的对象置入 StringBuffer 中
insert(int_start,String)	将指定的插入对象, 插入本字符串对象指定的开始位置中

【例 3.18】 字符串多重连接及插入方法的应用。

输入以下内容, 以 3_18strbuff.jsp 作为文件名保存:

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>字符串多重连接及插入方法的应用</title>
</head>
<body>
    <%
        String str1;
        StringBuffer str2;
        str1=new String("南京师范大学");
        str2=new StringBuffer();
        str2=str2.append(str1).append("欢迎您! ");
        out.print("str1 字符串对象为: "+str1+"<br>");
        out.print("str2 字符串为: " + str2.toString() + "<br>");
    %>
</body>
</html>
```

运行结果如图 3.18 所示。



图 3.18 字符串多重连接及插入方法的应用

3.2.3 常用日期/时间类

与日期和时间有关的类属于 `java.util` 包。

1. Date 类

`Date` 类可产生 `Date` 对象，并可指定对象内容为现在时间还是指定时间。

对象产生方式为：

`Date` 对象名称=`new Date()`;

`Date` 对象名称=`new Date(毫秒数)`;

方法说明如表 3.18 所示。

表 3.18 `Date` 类方法说明

方 法	说 明
<code>toString()</code>	返回现在时间，并以字符串类型显示
<code>getTime()</code>	返回 1970 年 1 月 1 日到现在的毫秒数，为长整型类型
<code>setTime(long)</code>	设置本对象自 1970 年 1 月 1 日起的毫秒数
<code>equals(Object)</code>	判断两个对象是否相等
<code>compareTo(Object)</code>	比较两个对象的大小

【例 3.19】`Date` 类方法的应用。

输入以下内容，以 `3_19date.jsp` 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*"%>
<html>
<head>
    <title>date 类方法的应用</title>
</head>
<body>
    <%
        Date d1=new Date();
        Date d2=new Date(86400000*20);
        long now=d1.getTime();
        out.print("d1 对象为: "+d1.toString()+"<br>");
        out.print("d2 对象为: "+d2.toString()+"<br>");
        out.print("<hr>");
        if(d1.compareTo(d2)>0){
            out.print("d1 比 d2 大! <br>");
        } else if(d1.compareTo(d2)==0){
            out.print("d1 和 d2 一样大! <br>");
        } else {
            out.print("d1 比 d2 小! <br>");
        }
        d2.setTime(now+86400000*3);
        //重新设置 d2 的时间
    %>
</body>
</html>
```

```
        out.print("三天后的时间为: " + d2.toString()+"<br>");  
    %>  
</body>  
</html>
```

运行结果如图 3.19 所示。

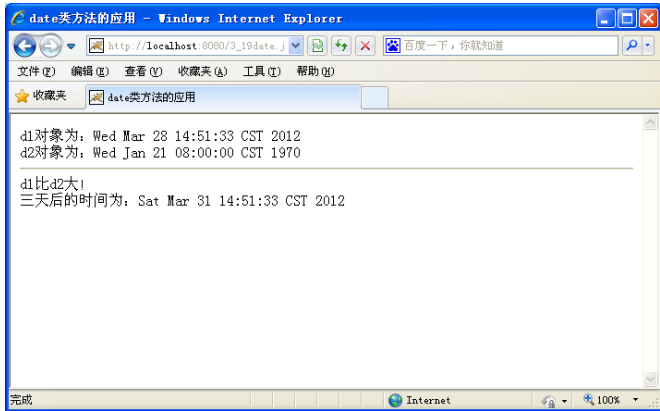


图 3.19 Date 类方法的应用

2. Calendar 类

Calendar 类方法说明如表 3.19 所示。

表 3.19 Calendar 类方法说明

方 法	说 明
equals(Object)	判断两个对象是否相等
getTime()	返回时间
set(int,int,int,int,int,int)	设置时间
get(Object)	按叙述式内容取出对象信息

说明：常用叙述式为 YEAR、MONTH、DAY_OF_MONTH、DAY_OF_WEEK、HOUR_OF_DAY、MINUTE、SECOND 等。

【例 3.20】Calendar 类方法的应用。

输入以下内容，以 3_20calen.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>  
<%@ page import="java.util.*"%>  
<html>  
<head>  
    <title>calendar 类方法的应用</title>  
</head>  
<body>  
    <%  
        Calendar c1;  
        int year,month,day,hour,min,sec;  
        c1=new GregorianCalendar();  
        out.print("现在时间是: "+c1.getTime().toString()+"<br>");
```



```

year=c1.get(Calendar.YEAR);
month=c1.get(Calendar.MONTH)+1;
day=c1.get(Calendar.DAY_OF_MONTH);
hour=c1.get(Calendar.HOUR_OF_DAY);
min=c1.get(Calendar.MINUTE);
sec=c1.get(Calendar.SECOND);
out.print("现在是: "+year+"年"+month+"月"+day+"日"+hour+"时"+min+"分"+sec+"秒");
out.print("<hr>");
//重设时间: 2012 年 12 月 21 日 15 时 14 分 35 秒
c1.set(2012,12-1,21,15,14,35);
out.print("时间重新设定后为: "+c1.getTime().toString()+"<br>");

%>
</body>
</html>

```

运行结果如图 3.20 所示。

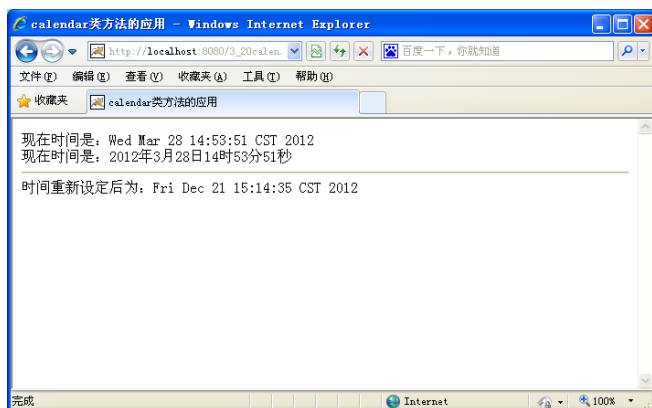


图 3.20 Calendar 类方法的应用

3.2.4 常用系统信息类

1. Package 类

Package 类属于 java.lang 包。方法说明如表 3.20 所示。

表 3.20 Package 类方法说明

方 法	说 明
toString()	返回字符串
getName()	返回程序名称
getSpecificationVersion()	返回版本信息
getSpecificationVendor()	返回厂商名称
getImplementationTitle()	返回包的名称
getPackage(String)	返回指定程序的包
getPackages()	返回所有包数组

【例 3.21】Package 类方法的应用。

输入以下内容，以 3_21pack.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<html>
<head>
    <title>package 类方法的应用</title>
</head>
<body>
    <%
        int i;
        out.print("Java 使用程序套件的名称: "+"<br>");
        Package pack[]=Package.getPackages();
        for(i=0;i<pack.length;i++)
            out.print("名称: "+pack[i].getName()+"<br>"+"内容为: "+pack[i].toString()+"<br>");
    %>
</body>
</html>
```

运行结果如图 3.21 所示。

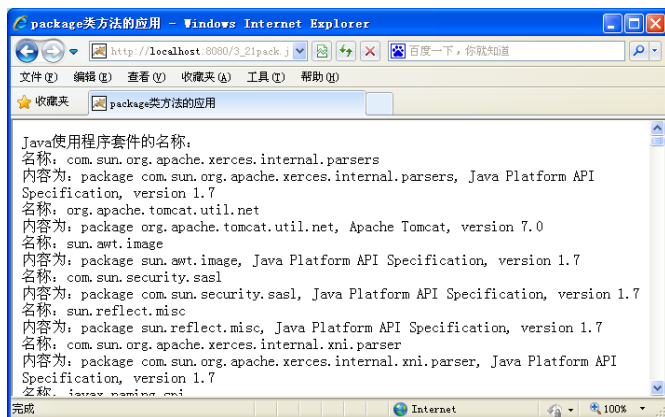


图 3.21 Package 类方法的应用

2. Runtime 类

Runtime 类属于 java.lang 包。方法说明如表 3.21 所示。

表 3.21 Runtime 类方法说明

方 法	说 明
getRuntime()	返回本对象的 Runtime 对象
totalMemory()	返回 JVM 全部内存
freeMemory()	返回当前可用内存
gc()	回收不用的内存
traceInstructions(boolean)	追踪 Java 命令
traceMethodCalls(boolean)	追踪 Java 方法
exec(String)	执行命令

【例 3.22】Runtime 类方法的应用。

输入以下内容，以 3_22run.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@page import="java.io.IOException"%>
<html>
<head>
    <title>runtime 类方法的应用</title>
</head>
<body>
    <%
        Runtime run=Runtime.getRuntime();
        out.print("总 内 存: "+run.totalMemory()/(1024*1024)+"M <br>");
        out.print("当前可用内存: "+run.freeMemory()/(1024*1024)+"M <br>");
        out.print("释放不用内存! <br>");
        run.gc();
        out.print("当前可用内存: "+run.freeMemory()/(1024*1024)+"M <br>");
        run.traceInstructions(true);
        run.traceMethodCalls(true);
        out.print("运行 eclipse ! <br>");
        try {
            run.exec("C:\\Program Files\\MyEclipse\\MyEclipse 9\\myeclipse.exe");
        } catch (IOException e) {
            e.printStackTrace();
        }
    %>
</body>
</html>
```

运行结果如图 3.22 所示。

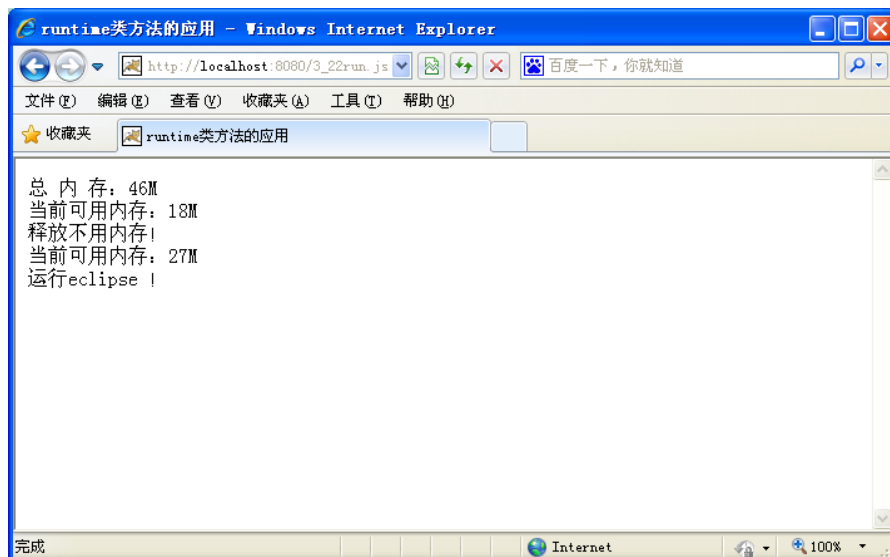


图 3.22 Runtime 类方法的应用

3. Hashtable 类

Hashtable 类属于 java.util 包。方法说明如表 3.22 所示。

表 3.22 Hashtable 类方法说明

方 法	说 明
put(Object,Object)	存对象
get(Object)	取对象
toString()	返回字符串
clear()	清空 Hash 表
size()	返回对象数
contains(Object)	返回 Hash 表中是否含有指定对象
containsKey(Object)	返回 Hash 表中是否含有指定对象的键值
remove(Object)	删除对象
keys()	返回键值, 返回值为 Enumeration 类型
elements()	返回对象值, 返回值为 Enumeration 类型
isEmpty()	判断 Hash 表是否为空

【例 3.23】Hashtable 类方法的应用。

输入以下内容, 以 3_23hash.jsp 作为文件名保存:

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*"%>
<html>
<head>
    <title>hashtable 类方法的应用</title>
</head>
<body>
    <%
        Hashtable table=new Hashtable(5,0.7F);
        table.put("1","南京大学");
        table.put("2","南京师范大学");
        table.put("3","东南大学");
        out.print("Hash 表中的对象数量为: "+table.size()+"<br>");
        out.print("Hash 表中的对象为: "+table.toString()+"<br>");
        out.print("Hash 表中的键值为[1]的对象是: "+table.get("1")+"<br>");
        out.print("Hash 表中是否包含[南京师范大学]的对象: "+table.contains("南京师范大学")+"<br>");

        out.print("Hash 表中是否包含键值为[4]的对象: "+table.containsKey("4")+"<br>");
        out.print("<hr>");
        out.print("删除键值为[3]的对象后: ");
        table.remove("3");           //删除 Hash 表中的第三个对象
        out.print("Hash 表中的对象数量为: "+table.size()+"<br>");
        out.print("Hash 表中的对象为: "+table.toString()+"<br>");
        out.print("Hash 表是否为空: "+table.isEmpty()+"<br>");
        Enumeration key=table.keys();    //将 Hash 表的键值设给对象 key
        Enumeration elem=table.elements(); //将 Hash 表的对象值赋给对象 elem
```

```

        out.print("<hr>");
        out.print("通过 keys 和 elements 方法获得 Hash 表中的对象键值和对象值<br>");
        while(elem.hasMoreElements())           //输出键值和对象值
        {
            out.print((String)key.nextElement()+" ");
            out.print((String)elem.nextElement()+"<br>");
        }
    }
%>
</body>
</html>

```

运行结果如图 3.23 所示。



图 3.23 Hashtable 类方法的应用

4. System 类

System 类属于 java.lang 包。方法说明如表 3.23 所示。

表 3.23 System 类方法说明

方 法	说 明
arraycopy(Object,int_start, Object,int,int_len)	复制数组
currentTimeMillis()	返回当前系统时间
exit(int)	终止 JVM 运行
getProperties()	返回系统属性对象
getProperty(String)	返回系统属性对象的值
identityHashCode(Object)	返回对象的 Hash 码
setProperty(Property)	设置系统属性
getSecurityManager()	返回安全管理器对象

【例 3.24】System 类方法的应用。

输入以下内容，以 3_24syst.jsp 作为文件名保存：

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.util.*"%>
<html>
<head>
    <title>system 类的应用</title>
</head>
<body>

```

```
<%  
    Properties ppt=System.getProperties();  
    out.print("本机操作系统目录: "+ppt.getProperty("user.home")+"<br>");  
    out.print("本机 java 程序的目录路径: "+ppt.getProperty("java.home")+"<br>");  
    out.print("本机 java class 的目录路径: "+ppt.getProperty("java.class.path")+"<br>");  
    out.print("用户名称: "+ppt.getProperty("user.name")+"<br>");  
    out.print("设置新用户! ");  
    System.setProperty("user.name","zaq");  
    out.print("新用户名称: "+ppt.getProperty("user.name")+"<br>");  
%>  
</body>  
</html>
```

运行结果如图 3.24 所示。

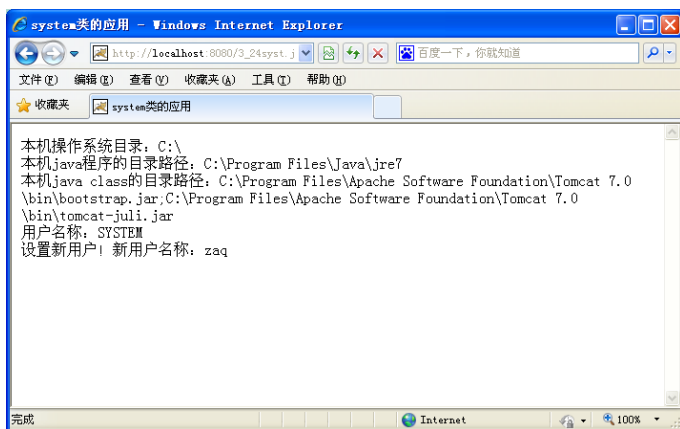


图 3.24 System 类的应用

3.3 JSP 编译指令

JSP 编译指令是给 JSP 引擎提供编译器指令信息的,其作用是设置 JSP 程序和由该 JSP 程序编译所生成的 Servlet 程序的属性。

语法格式:

```
<%@ 指令 指令的属性="属性值" %>
```

例如:

```
<%@ directive attribute= "value" %>  
<%@ directive  attribute1= "value1"  
                attribute2= "value2"  
                ...  
                attributeN= "valueN" %>
```

3.3.1 include 指令

JSP 语法中的 include 指令语句指示在 JSP 语句被解释的过程中包含一个静态文件,同时解析这个被包含文件中的 JSP 语句。在一个 JSP 页面中可以出现的 include 指令没有数量限制。

include 指令也可以被嵌套使用，在嵌套上也无限制。但是要注意的是，所有页面必须使用跟起始页面相同的脚本语言。

基本语法：

```
<% @ include file= "relativeURL" %>
```

其中，“relativeURL”指示被包含的文件的相对存储位置的 URL 地址。

另外，还要说明几点。

① 在 JSP 编译时插入一个包含文本或代码的文件，包括 JSP 文件、XHTML 文件和文本文件，或者只是一段 Java 代码。

② 包含文件的路径名一般来说是相对路径，不需要端口、协议和域名，如 error.jsp, /templates/onlinestore.html, /beans/calendar.jsp 等。

③ 包含文件中不能使用<html>,</html>,<body>,</body>标记,因为这将影响原 JSP 文件中同样的标记而导致错误。

【例 3.25】include 指令的应用。

输入以下内容，以 3_25include.jsp 作为文件名保存：

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
    <title>include 指令的应用</title>
</head>
<body>
    <font color="blue" >
        <h1> include 指令示例 </h1>
        本机的 IP 地址是：
        <%@ include file="3_25BeenInc.jsp" %>
    </font>
</body>
</html>
```

引入文件 3_25BeenInc.jsp 的代码如下：

```
<%= request.getRemoteAddr() %>
```

将这两个文件都复制到 C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ROOT 目录下，运行结果如图 3.25 所示。



图 3.25 include 指令的应用

3.3.2 page 指令

page 指令的语法规则如下：

```
<%@ page
[ language="java" ]
[ extends="package.class" ]
[ import="{ package.class | package.*}, ..." ]
[ session="true | false" ]
[ buffer="none | 8kb | sizekb" ]
[ autoFlush="true | false" ]
[ isThreadSafe="true | false" ]
[ info="text" ]
[ errorPage="relativeURL" ]
[ contentType="mimeType [; charset=characterSet ]" | "text/html; charset=ISO-8859-1" ]
[ isErrorPage="true | false" ]
%>
```

可以看出，page 指令包含了多个“属性-数值”对，而且每条 page 指令可以包含其中的一条或若干条内容。下面将逐一介绍这些属性的意义。

language="java"：声明在 JSP 页面中使用脚本语言的种类，默认为 Java。

extends="package.class"：指明生成的 Servlet 的父类，但必须慎重使用，默认为 `HttpJspBase`。

import="{ package.class | package.*}, ..."：需要导入的 Java 类或包的列表。默认导入的 Java 类包含 `java.lang.*`、`java.servlet.*`、`javax.servlet.jsp` 和 `javax.servlet.http.*`。其他任何将要使用的类都必须用 import 属性包含进来。

session="true | false"：设定 JSP 文件中是否需要使用 HTTP Session，默认值为 `true`。

buffer="none | 8kb | sizekb"：指明该 JSP 程序中 `out` 内置对象的缓冲区 `buffer` 的大小，默认值因服务器而异，但通常为 8kb。如果设置了这一属性，则缓冲大小不会小于设置的数值。

autoFlush="true | false"：设置如果 `buffer` 溢出，是否需要强制输出，默认值为 `true`。

isThreadSafe="true | false"：设置文件生成的 Servlet 是否能多线程使用。如果属性为真，则 JSP 可以同时处理多个请求；如果为假，则一次只能处理一个请求。

info="text"：设定一个文本，该文本能够使用 `Servlet.getServletInfo()` 方法取回。

errorPage="relativeURL"：设置一个出错页，用于处理本 JSP 页面没有捕捉到的异常事件。“relativeURL”用于定义处理异常事件的 JSP 文件。

contentType 属性定义输出为 MIME 类型，默认为 `text/html`；默认字符集为 `ISO-8859-1`。

isErrorPage="true | false"：设置此页是否能作为另一个 JSP 网页的出错页，默认值为 `false`。

例如，导入 Java 包 `java.util.*` 和 `java.lang.*` 的指令如下：

```
<%@ page import="java.util.*, java.lang.*" %>
```

设置页面的缓冲区为 24kb，自动刷新为 `true` 的指令如下：

```
<%@ page buffer="24kb" autoFlush="true" %>
```

设置用于处理异常错误的 JSP 文件的指令如下：

```
<%@ page errorPage="error.jsp" %>
```

page 指令提供了 JSP 页面的属性。在 page 指令中定义的属性用于该 JSP 页面，以及所有通过 `include` 指令或 `<jsp:include>` 动作包含的静态文件，但是不能用于动态文件。

在之前的诸多例子中，几乎每个源文件的开头都有如下代码：

```
<%@ page contentType="text/html; charset=GB2312" %>
```

这是在设置输出为 MIME 类型，并且支持中文汉字编码标准 GB2312。只有进行了这样的设置，网页才能在绝大多数情况下正常显示。

3.3.3 taglib 指令

在 JSP 中有时会用到标签，这时就要用到 taglib 指令，其语法格式如下：

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

其中 uri = " tagLibraryURI " 指明标签库文件的存放位置，而 prefix = " tagPrefix " 则表示该标签使用时的前缀。例如，如果开发中要用到 Struts 2 框架，则必须进行如下声明：

```
<%@ taglib uri = "/struts-tags" prefix = "s"%>
```

才能在程序代码中引用 Struts 2 的标签。

3.4 JSP 动作元素

JSP 动作元素又叫 JSP 操作指令，与 JSP 编译指令不同，它是在客户端请求时才动态地被执行的。每次有客户端请求到达时，可能都会被重新执行一次，而 JSP 编译指令在编译时即被 JSP 引擎执行。

JSP 动作元素比较多，目前最新的 JSP 技术规范支持下列元素：<jsp:param>、<jsp:include>、<jsp:useBean>、<jsp:setProperty>、<jsp:getProperty>、<jsp:forward>、<jsp:plugin>等。

JSP 动作元素用来控制 JSP 引擎的行为，可以动态地插入文件、重用 JavaBean 组件、导向另一个页面等。

3.4.1 <jsp:param>

<jsp:param>的语法规则如下：

```
<jsp:param name="paramName" value="paramValue"/>
```

例如：

```
<jsp:param name="username" value="jack"/>
```

上面的操作就是将 jack 的值和 username 对应起来，从而使 jack 和 username 两者相关联。

<jsp:param>通常与<jsp:include>、<jsp:forward>或<jsp:plugin>等一起使用。在独立于其他操作使用时，<jsp:param>动作没有作用。

3.4.2 <jsp:include>

<jsp:include>的语法规则如下：

```
<jsp:include page=" { relativeURL | <%= expression %> } " flush="true" />
```

或

```
<jsp:include page=" { relativeURL | <%= expression %> } " flush="true" >
```

```
<jsp:param name="paramName" value="{ paramValue | <%= expression %>}" />
```

```
</jsp:include>
```

`<jsp:include>`可以向一个对象提出请求，并可以将结果包含在一个 JSP 文件中。其中参数 `page="{relative URL | <%= expression %>}"` 为相对路径，或者代表相对路径的表达式。参数 `flush` 必须使用 `flush="true"`，不能使用 `flush="false"`，因为在 JSP1.1 规范中，`flush="false"` 是不允许的。

在 `<jsp:param name="paramName" value="{ paramValue | <%= expression %>}" />` 中，使用 `<jsp:param>` 指令允许传递一个或多个参数给被包含到主 JSP 程序中的动态程序，能在一个 JSP 程序中使用多个 `<jsp:param>` 指令来传递多个参数给被包含的目标程序。

`<jsp:include>`可以将静态的 XHTML、服务器程序的输出结果，以及来自其他 JSP 的输出包括到当前页面中。它使用相对的 URL 来调用资源。

例如，包含普通的 XHTML 文件：

```
<jsp:include page="hello.html" />
```

使用相对路径：

```
<jsp:include page="/index.html" />
```

包含动态 JSP 文件：

```
<jsp:include page="scripts/login.jsp" />
```

向被包含的程序传递参数：

```
<jsp:include page="scripts/login.jsp" >
```

```
<jsp:param name="username" value="jack" />
```

```
</jsp:include>
```

`<jsp:include>`指令允许包含动态文件和静态文件，这两种方式的结果是不同的：如果是静态文件，那么仅仅是把包含文件的内容加到 JSP 文件中；而如果是动态文件，那么被包含文件也会被 JSP 编译器执行。一般不能从文件名上判断一个文件是动态的还是静态的，比如 `hello.jsp` 就有可能只包含一些静态的 XHTML 标记，而不需要执行 Java 脚本。

【例 3.26】`<jsp:include>`动作元素的应用。

输入以下内容，以 `3_26jsp_include.jsp` 作为文件名保存：

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<html>
<head>
    <title>include 动作元素的应用</title>
</head>
<body>
    <h2>带参数的引用</h2>
    <jsp:include page="3_26two.jsp" flush="true">
        <jsp:param name="a1" value="Hello" />
        <jsp:param name="a2" value="World!" />
    </jsp:include>
</body>
</html>
```

引入的页面 `3_26two.jsp` 的代码如下：

```
<%
    String a1=request.getParameter("a1");
    String a2=request.getParameter("a2");
    out.println(a1);
    out.println(a2);
%>
```

运行结果如图 3.26 所示。

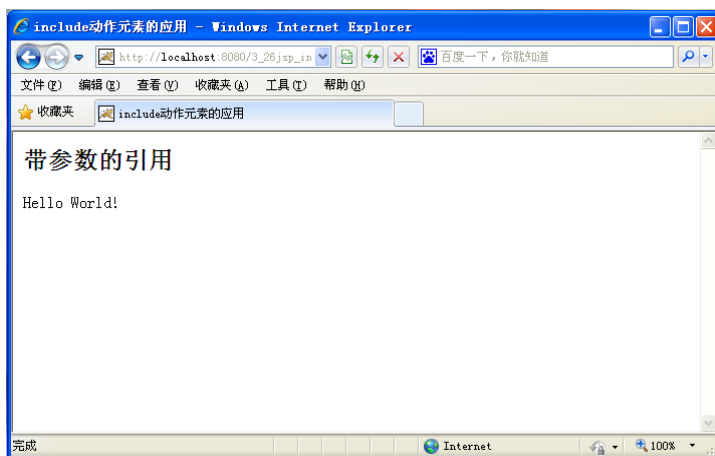


图 3.26 include 动作元素的应用

3.4.3 <jsp:useBean>

<jsp:useBean>的语法规则如下:

```
<jsp:useBean id="name" class="classname" scope="page | request | session | application" typeSpec />
```

语法参数说明如下。

- ① id 属性用来设置 JavaBean 的名称, 利用此 id, 可以识别在同一个 JSP 程序中使用的不同 JavaBean 组件实例。
- ② class 属性用于指定 JavaBean 对应的 Java 类名查找该 JavaBean 的路径。
- ③ scope 属性指定 JavaBean 对象的作用域。scope 的值可能是 page,request,session 及 application。
- ④ typeSpec 可能是如下的四种形式之一:
 - class="className";
 - class="className" type="typeName";
 - beanName="beanName" type="typeName";
 - type="typeName"。

<jsp:useBean>的功能首先是初始化一个“class”属性所指定的 Bean 类的实体, 并将该属性实体命名为“id”属性所指定的值。但是, 如果系统中已经存在相同的“id”和“scope”属性的 Bean 实体, 则该动作将不再初始化新的实体, 而是直接使用已经存在的 Bean 对象。

通过<jsp:useBean>动作指令在 JSP 页面中声明了 Bean 类实体后, 就可以使用<jsp:setProperty>或<jsp:getProperty>指令设置或读取 Bean 的属性。同时, 也可以使用 JSP 脚本程序或表达式直接调用 Bean 对象的公有方法。

【例 3.27】<jsp:useBean>动作元素的应用。

输入以下内容, 以 3_27bean.jsp 作为文件名保存:

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
```

```
<title>usebean 动作元素的应用</title>
</head>
<body>
  <jsp:useBean id="test" scope="page" class="test.TestBean" />
  <%
    test.setString("南京师范大学");
    String str=test.getStringValue();
    out.print(str);
  %>
</body>
</html>
```

其中 TestBean.java 的代码如下:

```
package test;
public class TestBean{
    private String str = null;
    public TestBean(){ } //构造函数
    public void setString(String value)
    {
        str = value;
    }
    public String getStringValue()
    {
        return str;
    }
}
```

将 TestBean.java 文件编译后的.class 文件 TestBean.class 放在 Tomcat 安装路径的\webapps\ROOT\WEB-INF\classes\test 目录下 (test 目录要自己建立)。运行结果如图 3.27 所示。

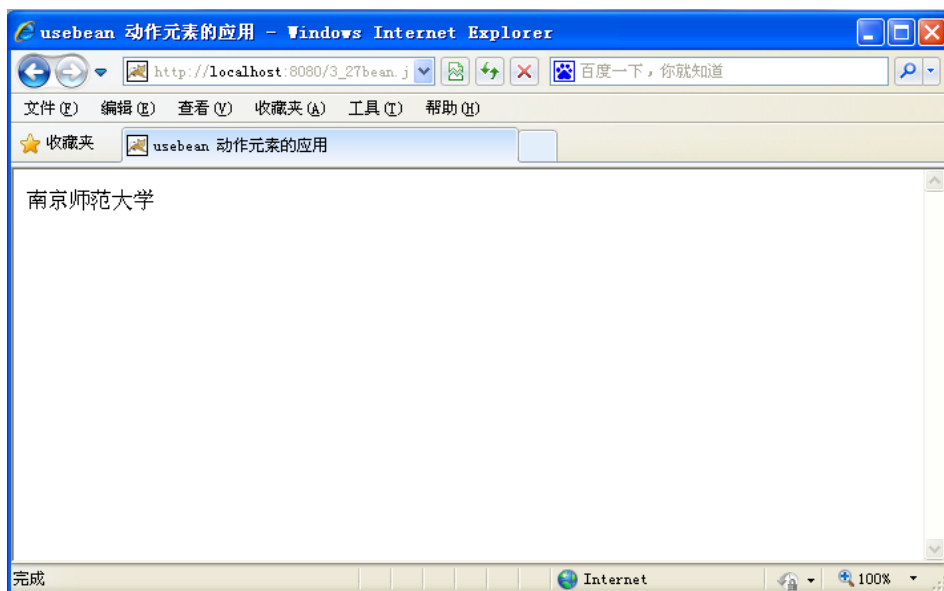


图 3.27 useBean 动作元素的应用

3.4.4 <jsp:setProperty>与<jsp:getProperty>

1. <jsp:setProperty>

<jsp:setProperty>的语法规则如下:

```
<jsp:setProperty name="BeanName "
{
    property="*" |
    property="propertyName" [ param="parameterName" ] |
    property="propertyName" value="propertyValue "
}
/>
```

语法参数说明如下。

① name 属性指定了目标 Bean 对象。

② property 属性指定了要设置 Bean 的属性名。如果 property 的值是 “*”，则 “request” 对象中的所有与 Bean 属性同名的参数值都将传递给相应属性的赋值方法。Bean 中的属性名与 Request 中的参数名必须相同。

③ value 属性用来指定 Bean 属性的值。这个值可以是一个 String 常量，也可以是一个表达式。value 的字符串数据将会自动地转换为相应的 Bean 属性的类型。

<jsp:setProperty>将字符串类型转换为其他类型的方法如下:

```
boolean (或 Boolean) —— java.lang.Boolean.valueOf(String);
byte (或 Byte) —— java.lang.Byte.valueOf(String);
char (或 Character) —— java.lang.Character.valueOf(String);
double (或 Double) —— java.lang.Double.valueOf(String);
float (或 Float) —— java.lang.Float.valueOf(String);
int (或 Integer) —— java.lang.Integer.valueOf(String);
long (或 Long) —— java.lang.Long.valueOf(String);
```

④ param 属性指定了从 “request” 对象的某一参数取值以设置 Bean 的同名属性，即要将其值赋给一个 Bean 属性的 HTTP 请求的参数名称。

根据 JSP 规范，如下代码都是合法的:

```
<jsp:setProperty name="TestBean" property="*" />
<jsp:setProperty name="TestBean" property="username" />
<jsp:setProperty name="TestBean" property="username" value="jack" />
```

2. <jsp:getProperty>

<jsp:getProperty>的语法规则如下:

```
<jsp:getProperty name="BeanName" property="PropertyName" />
```

其中属性 name 是 JavaBean 实例的名称，property 是要显示的属性的名称。

根据语法规则，如下代码是合法的:

```
<jsp:useBean id="test" scope="page" class="test.TestBean" />
<h1>
    Get of string :<jsp:getProperty name="test" property="StringValue" />
</h1>
```

<jsp:getProperty>可以获取 Bean 的属性值。它从 Bean 的属性中取出值并转化成字符串，然后放在输出缓冲区。<jsp:getProperty>的使用方法和<jsp:setProperty>相似。

3. 示例

【例 3.28】setProperty 和 getProperty 动作元素的应用。

输入以下内容，以 3_28setgetpro.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=GB2312" %>
<html>
<head>
    <title>setProperty 和 getProperty 动作元素的应用</title>
</head>
<body>
    <jsp:useBean id="testbean" class="test.SetGetBean" />
    <jsp:setProperty name="testbean" property="string1" value="123"/>
    <jsp:setProperty name="testbean" property="string2" value="456"/>
    The value of string1 is: <jsp:getProperty name="testbean" property="string1"/><br>
    The value of string2 is: <jsp:getProperty name="testbean" property="string2"/>
</body>
</html>
```

其中 SetGetBean.java 的代码如下：

```
package test;
public class SetGetBean{
    private String string1 = null;
    private String string2 = null;
    public SetGetBean (){}
    public void setString1(String value)
    {
        string1 = value;
    }
    public void setString2(String value)
    {
        string2 = value;
    }
    public String getString1()
    {
        return string1;
    }
    public String getString2()
    {
        return string2;
    }
}
```

将 SetGetBean.java 文件编译后的.class 文件 SetGetBean.class 放在 Tomcat 安装路径的 \webapps\ROOT\WEB-INF\classes\test 目录下。运行结果如图 3.28 所示。

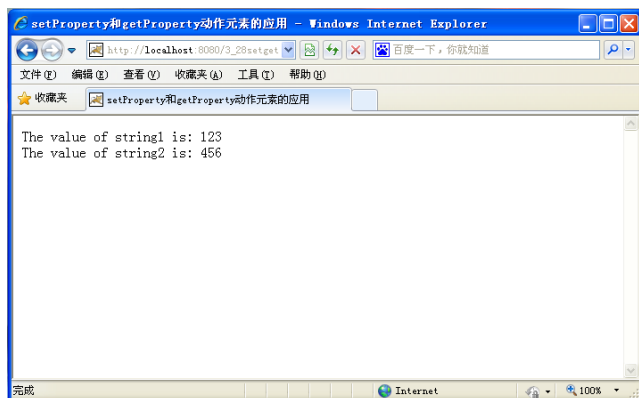


图 3.28 setProperty 和 getProperty 动作元素的应用

3.4.5 <jsp:forward>

<jsp:forward>的语法规则如下：

```
<jsp:forward page=" { relativeURL | <%= expression %> } " />
```

或

```
<jsp:forward page=" { relativeURL | <%= expression %> } ">
```

```
<jsp:param name="paramName" value="{ paramValue | <%= expression %>}" />
```

```
</jsp:forward>
```

<jsp:forward>可以重定向到一个 XHTML 文件、JSP 文件或一个程序段。<jsp:forward>动作把用户的请求转到另外的页面进行处理。<jsp:forward>标记只有一个属性 page，它指定要转发资源的相对 URL。page 的值既可以直接给出，也可以在请求的时候动态计算。

例如：

```
<jsp:forward page="/utils/errorReporter.jsp" />
```

```
<jsp:forward page="<%= someJavaExpression %>" />
```

3.4.6 <jsp:plugin>

<jsp:plugin>的语法规则如下：

```
<jsp:plugin type="bean | applet" code="className" codebase="classFileDirectoryName"
```

```
  [ name="instanceName" ]
```

```
  [ archive="URIToArchive ,..." ]
```

```
  [ align="bottom | top | middle | left | right" ]
```

```
  [ height="displayPixels" ]
```

```
  [ width="displayPixels" ]
```

```
  [ hspace="leftRightPixels" ]
```

```
  [ vspace="topBottomPixels" ]
```

```
  [ jreversion="JREVersionNumber | 1.2 " ]
```

```
  [ nspluginurl="URLToPlugin" ]
```

```
  [ iepluginurl="URLToPlugin" ]>
```

```
  [ <jsp:params>
```

```
    [ <jsp:params name="paramName" value="{ parameterValue | <%= expression %>}" />]+</jsp:params>
```

```
[ <jsp:fallback> text message for user </jsp:fallback> ]  
</jsp:plugin>
```

语法参数说明如下。

- ① **type**: 指定被执行的 Java 程序的类型是 JavaBean 还是 Java Applet。这个属性没有默认值, 所以必须确定该属性的值。
- ② **code**: 指定将会被浏览器的 JVM 执行的 Java Class 的名字, 必须以 .class 结尾命名。
- ③ **codebase**: 指定将会被执行的 Java Class 文件所在的目录或路径, 默认值为调用</jsp:plugin>指令的 JSP 文件的目录。
- ④ **name**: 确定这个 JavaBean 或 Java Applet 程序的名字, 它可以在 JSP 程序的其他地方被调用。
- ⑤ **archive**: 表示包含对象 Java 类的.jar 文件。
- ⑥ **align**: 对图形、对象、applet 等进行定位, 可以选择的值为 bottom,top,middle,left 和 right 这五种。
- ⑦ **height** 和 **width**: JavaBean 或 Java Applet 将要显示出来的高度、宽度的值, 此值为数字, 单位为像素。
- ⑧ **hspace** 和 **vspace**: JavaBean 或 Java Applet 显示时在浏览器显示区左右、上下所需留下的空间, 单位为像素。
- ⑨ **javaversion**: JavaBean 或 Java Applet 被浏览器正确运行所需要的 Java 运行时环境 (Java Runtime Environment, JRE) 的版本, 默认值是 1.2。
- ⑩ **pluginurl**: 可以为 Netscape Navigator 用户提供下载 JRE 插件的地址。此值为一个标准的 URL, 如 <http://www.njnu.edu.cn>。
- ⑪ **iepluginurl**: IE 用户下载 JRE 的地址。此值为一个标准的 URL, 如 <http://www.njnu.edu.cn>。
- ⑫ **<jsp:params>** 和 **</jsp:params>**: 使用<jsp:params>操作指令, 可以向 JavaBean 或 Java Applet 传送参数和参数值。
- ⑬ **<jsp:fallback>** 和 **</jsp:fallback>**: 该指令中间的一段文字用于 Java 插件不能启动时显示给用户, 如果插件能够正确启动, 而 JavaBean 或 Java Applet 的程序代码不能找到并被执行, 那么浏览器将会显示出错信息。

例如:

```
<jsp:plugin  
  type="applet"  
  code="Test.class"  
  codebase="/example/jsp/applet "  
  height="180"  
  width="160"  
  javaversion="1.2">  
  <jsp:params>  
    <jsp:params name="test" value="TsetPlugin" />  
  </jsp:params>  
  <jsp:fallback>  
    <p> To load apple is unsuccessful </p>  
  </jsp:fallback>  
</jsp:plugin>
```


3.5 上机练习

1. 根据本章实例，设计网页，文件名为 3_4con.jsp，显示如图 3.29 所示（与图 3.4 相同）。

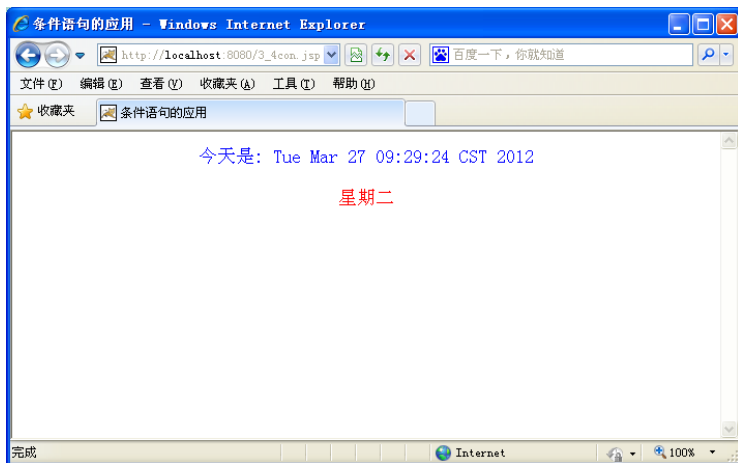


图 3.29 条件语句的应用

修改上述代码，使其能够计算后天是星期几。

2. 根据本章实例，设计网页，文件名为 3_6rep2.jsp，页面显示如图 3.30 所示（与图 3.6 相同）。

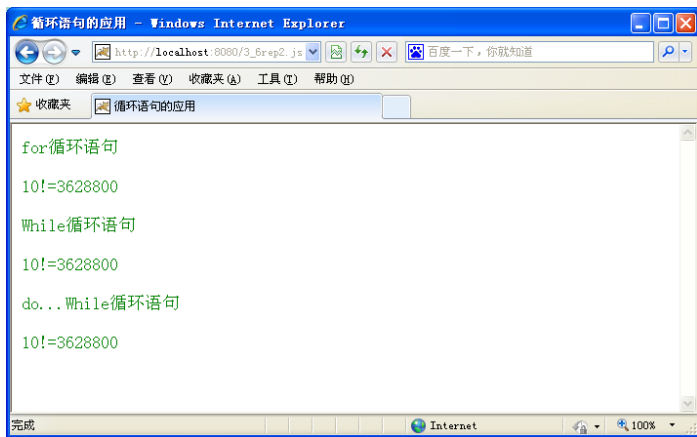


图 3.30 循环语句的应用

- (1) 修改上述代码，使用三种循环方式的一种，使其能够计算 $1 \times 2 \times 3 \dots n$ 。
- (2) 修改上述代码，使用三种循环方式的一种，使其能够输入起点、终点和步长。
- (3) 修改上述代码，使用三种循环方式的一种，使其能够同时计算 $1 \sim n$ 的和与乘积，通过单选按钮进行选择是计算求和还是计算阶乘。
- (4) 将计算 $1+2+3 \dots +n$ 和计算 $1 \times 2 \times 3 \dots \times n$ 分别编成函数和过程，通过调用函数和过程得到计算结果并显示。

第 4 章

前端页面开发技术

2005 年以后，互联网进入 Web 2.0 时代，各种类似桌面软件的 Web 应用大量涌现，网站的前端由此发生了翻天覆地的变化。网页不再只承载单一的文字和图片，各种富媒体让网页的内容更加生动，网页上软件化的交互形式为用户提供了更好的使用体验，这些都是基于前端技术实现的。

Web 前端开发技术包括三个要素：HTML/XHTML、CSS 和 JavaScript。其中，前两者用于网页设计（第 1 章已有介绍），这里重点讲 JavaScript 及其应用。

4.1 JavaScript 基础

4.1.1 脚本语言简介

脚本语言就是可以和 HTML/XHTML 混合使用的语言，JavaScript 和 VBScript 是两种最为主流的脚本语言，其中 VBScript 只有微软公司的 IE 浏览器才能完全支持，而 JavaScript 在任何浏览器上都可运行，是 JSP 前端开发的首选。

JavaScript 作为一种高级脚本语言，具有以下三大优点。

- ① JavaScript 采用在 HTML/XHTML 文本中嵌入小程序段的方式，开发过程非常简单，并且提高了响应速度。
- ② JavaScript 可以直接对用户或客户的输入做出响应，而不需要经过 Web 服务器的程序，减少了客户浏览器与服务器之间的通信量，提高了速度。
- ③ JavaScript 是一种与平台无关的解释性语言，独立于操作系统，只要计算机能运行浏览器且该浏览器支持 JavaScript，就可执行 JavaScript 脚本程序。

脚本语言的语法与一般的编程语言并没有什么不同，只是减少了一些可能会对 Web 浏览用户产生伤害的功能。在 Web 中应用 JavaScript 可以大大加强 Web 页的交互性，使 Web 页上显示的文本信息“动”起来，或者加入一些动画，使 Web 页更加灵活。它嵌入在标准的 JSP 源文件中，在用户端和服务器端均可执行。

虽然 JavaScript 具有交互性，但它本身不具备将数据传回服务器的能力，也没有访问服务器上数据的能力，故要与服务器打交道，还要通过 JSP 服务器对象（见本书第 5 章）。

4.1.2 网页中的 JavaScript

在网页中引入 JavaScript，只需加入<script>元素，然后再设置所用语言即可。

【例 4.1】通过 JavaScript 输出一句话。

在记事本中编辑如下代码：

```
<html>
<body>
  <meta http-equiv="content-type" content="text/html; charset=gb2312">
  <script language="javascript">
    document.write("欢迎来到易睿得工作室，相信您会找到您所需要的知识!!! ");
  </script>
</body>
</html>
```

编辑完后，以文件名 4_1welcome.html 保存，程序运行结果如图 4.1 所示。



图 4.1 通过 JavaScript 输出一句话

在上述代码中，使用了一对<script>和</script>元素，元素中的 language 属性设置为 JavaScript，这是因为脚本使用的是 JavaScript 语言。事实上，如果省略了 language 属性的设置，IE 浏览器也不会出错。<script>和</script>元素中的脚本是使用 JavaScript 语言编写的，只有一条语句“document.write("欢迎来到易睿得工作室，相信您会找到您所需要的知识!!!")”，该语句的功能是向浏览器输出括号中的字符串。

4.1.3 基本语法

JavaScript 脚本语言同其他语言一样，有自身的基本数据类型、表达式、运算符及程序的基本框架结构。下面将介绍 JavaScript 常见的语法知识。

1. 数据类型

在 JavaScript 语言中，常见的数据类型如表 4.1 所示。

表 4.1 JavaScript 中的数据类型

名 称	类 型	含 义
number	数值型	该类型包含整数和浮点数。整数可以为正整数或负整数，浮点数可以包括小数点，如“5.33”或“7E-2”
string	字符串型	字符串数据应加上单引号或双引号
boolean	布尔型	可以为 true 或 false 两个值
object	对象型	该类型是 JavaScript 的重要组成部分

与其他编程语言相同，JavaScript 中的数据也分为常量和变量。

2. 常量

JavaScript 中的常量分为整型常量、实型常量、布尔常量、字符型常量、空值和转义符几种。

- 整型常量：可以使用十六进制、八进制和十进制数表示。
- 实型常量：由整数部分加小数部分表示，如 3.14、0.618 等；也可以使用科学计数法或标准方法表示，如 1e3、4e5 等。
- 布尔常量：只有 true 和 false 两种形式，主要用来说明或代表一种状态或标志。
- 字符型常量：是用单引号 “'” 或双引号 “”” 括起来的一个或几个字符。
- 空值 null：当引用没有赋值的常量时会返回一个 null 值。
- 转义符：当要引用某些特殊字符时，可以使用 “\”。例如，\n 表示换行，\\ 表示 “\”，\" 表示 “””。

3. 变量

JavaScript 对变量的数据类型要求并不严格（这一点和其他语言不同）。在 JavaScript 中，可以用 var 关键字声明变量，而不指定变量类型，例如下面的语句：

```
var a;
```

在这种情况下，该变量还不知是哪种类型，赋值时才清楚，例如：

```
var a;
```

```
a = 5;
```

为变量 a 赋予 int 型值 5，它就是 int 型了。

当然，也可以在定义变量时直接赋值，例如：

```
var a = "25";
```

为变量 a 赋予字符串类型值“25”。

在 JavaScript 中，变量也可以不做声明，使用时根据数据类型来确定其变量的类型，例如下面的语句：

```
i = 5;
```

```
j = "abc";
```

```
k = true;
```

```
x = 0.618;
```

JavaScript 中的变量与 Java 类似，也有全局变量和局部变量之分。全局变量在所有函数体之外，对所有函数均可见；而局部变量定义在函数体内部，只在该函数中可见。

变量命名需要遵守以下 5 个规则。

- ① 变量命名必须以一个英文字母或下画线开头，也就是变量名的第 1 个字符必须是 A 到 Z（或 a 到 z）之间的字母，或是 “_”。
- ② 变量名长度在 0~255 字符之间。
- ③ 除了首字符，其他字符可以使用任何字母、数字或下画线，但不能使用空格。
- ④ 不能使用 JavaScript 的保留字。
- ⑤ 不能使用 JavaScript 的运算符。

变量主要用于存取数据及提供存放信息的容器。

4. 数组

数组由一组数值按照顺序排列在一起，并放在同一个变量中，而每个数值都可以通过索引得到。例如，声明含有两个数的一维数组的代码如下：

```
var arrUserInfo = new Array(2)
```

在上述代码中，第 1 个数的索引下标是 0，第 2 个数的索引下标是 1。

声明数组时，使用 new 和 Array 关键字。new 代表建立一个新的对象，Array 是 JavaScript 内置的一个对象，由于 JavaScript 区分大小写，因此 Array 的首字母必须是大写的。

5. 运算符

JavaScript 运算符可分为三类：算术运算符（见表 4.2）、比较运算符（见表 4.3）和逻辑运算符（见表 4.4）。

表 4.2 算术运算符

运 算 符	功 能
+	加
-	减
*	乘
/	除
%	取模
	按位或
&	按位与
<<	左移
>>	右移
>>>	右移、零填充
-	取反
~	取补
++	递增 1
--	递减 1

表 4.3 比较运算符

运 算 符	功 能
<	小于
>	大于
<=	小于等于
>=	大于等于
==	等于
!=	不等于

表 4.4 逻辑运算符

运 算 符	功 能
!	取反
&=	与后赋值
&	逻辑与
=	或后赋值
	逻辑或
^=	异或之后赋值
^	逻辑异或
?:	三目操作符
	或
==	等于
!=	不等于

4.1.4 常用语句

在 JavaScript 语言中，通常采用一个或多个关键字完成给定的程序。语句可以非常简单，如一个函数名称、一个退出语句；也可以非常复杂，如声明一组要反复执行的名称。下面将介绍 JavaScript 语言中标准的常用语句。

1. 函数定义语句

JavaScript 的函数相当于 Java 语言中的方法，用于完成所需要的功能。通常在写一个复杂程序时，总是根据所完成功能的不同，将程序划分为一些相对独立的部分，每个部分由一个函数来完成。从而使各部分独立，任务单一，程序清晰、易懂。

JavaScript 函数定义格式如下：

```
function 函数名称(参数)
{
    函数执行部分
    return 表达式
}
```

上述代码中，`return` 语句表示函数的返回值，通过 JavaScript 函数格式定义一个函数的代码如下：

```
function easybooks()
{
    alert("欢迎来到易睿得！");
    return 表达式
}
```

2. 条件语句

条件语句通过 `if...else` 来完成程序流程块中的分支功能，具体格式如下：

```
if(条件)
{
    执行语句 1;
}
else
{
    执行语句 2;
}
```

在上述代码中，如果条件成立，则执行语句 1，否则执行语句 2。

3. 分支语句

分支语句 `switch` 是根据一个表达式取值的不同而采用不同的处理方法，具体格式如下：

```
switch(表达式)
{
    case 1: 执行语句 1;
    case 2: 执行语句 2;
    case 3: 执行语句 3;
    ...
}
```

在上述代码中，如果表达式的值与 `case` 值都不匹配，将执行省略号处的语句。

4. 循环语句

在 JavaScript 语言中，循环语句包含 `for` 语句、`for...in` 语句和 `while` 语句。下面将分别进行介绍。

(1) `for` 语句

`for` 语句的功能是只要循环条件成立，就反复执行循环体中的语句，具体格式如下：

```
for(变量初始化; 条件; 更新变量)
{
    执行语句;
}
```

(2) `for...in` 语句

`for...in` 语句与 `for` 语句相似，不同的是 `for...in` 循环的范围是一个对象的所有属性或一个数组中的所有元素，具体格式如下：

```
for(变量 in 对象或数组)
{
    执行语句;
}
```

(3) `while` 语句

`while` 语句中的条件一旦成立，则循环下去，直到条件不再成立为止，具体格式如下：

```
while(条件)
{
    执行语句;
}
```

4.1.5 对象

JavaScript 是基于对象的语言，每个对象都有自己的方法，而且 JavaScript 语言本身具有很多对象，下面将介绍一些常用的对象。

1. 时间对象：Date

Date 对象的主要作用是获取当前的系统时间，使用该对象必须使用关键字 `new` 来创建。例如下面的代码：

```
var date = new Date();
```

Date 对象的方法如表 4.5 所示。

表 4.5 Date 对象的方法

名 称	含 义	名 称	含 义
<code>getFullYear()/setFullYear()</code>	获取或赋值当前的年份	<code>getMonth()/setMonth()</code>	获取或赋值当前的月份
<code>getDate()/setDate()</code>	获取或赋值当前的日期	<code>getDay()/setDay()</code>	获取或赋值当前的星期
<code>getHours()/setHours()</code>	获取或赋值当前的小时	<code>getMinutes()/setMinutes()</code>	获取或赋值当前的分钟
<code>getSeconds()/setSeconds()</code>	获取或赋值当前的秒	<code>getTime()/setTime()</code>	获取或赋值当前的时间（以毫秒为单位）

2. 数学对象：Math

Math 对象可以用来处理各种数学运算，其内置方法定义了各种数学运算，可以直接调用。Math 对象的方法如表 4.6 所示。

表 4.6 Math 对象的方法

名 称	含 义	名 称	含 义
abs(x)	返回 x 的绝对值	acos(x)	返回 x 的反余弦值
asin(x)	返回 x 的正弦值	atan(x)	返回 x 的反正切值
ceil(x)	返回大于或等于 x 的最小整数	cos(x)	返回 x 的余弦值
exp(x)	返回 e 的 x 次方	floor(x)	返回小于或等于 x 的最大整数
max(x,y)	返回 x、y 中的最大值	min(x,y)	返回 x、y 中的最小值
pow(x,y)	返回 x 的 y 次方	round(x)	返回 x 的整数部分
sin(x)	返回 x 的正弦值	sqrt(x)	返回 x 的平方根
tan(x)	返回 x 的正切值		

3. 字符串对象：String

String 是字符串对象，也是使用较多的对象，该对象只有一个属性：length 属性表示字符串中包含的字符数目。

String 对象的方法如表 4.7 所示。

表 4.7 String 对象的方法

名 称	含 义	名 称	含 义
big()	设置字符串为大字体	small()	设置字符串为小字体
italics()	设置字体为斜体	fixed()	设置固定字体
bold()	设置字体为粗体	substring(start,end)	获取自 start 到 end 的子串
toUpperCase()	转换字符串为大写	toLowerCase()	转换字符串为小写
fontSize(size)	设置字体的大小，参数 size 为整数，数越大字体就越大		
fontcolor(color)	设置字体的颜色，参数 color 可以使用 blue、red 等表示，也可以使用 ff0233 等 6 位十六进制数表示		
indexOf(char,start)	在安装和检验阶段查找串中从 start 处第 1 个出现的 char 字符，并返回其位置		

4.1.6 事件

在基于 Windows 平台的程序设计中，事件（event）是一个很重要的概念。所谓事件就是由某个对象发出的消息，这个消息标志着某个特定的行为发生，或某个特定的条件成立。例如，单击鼠标、单击按钮或者打开窗口时，都会触发相应的事件。下面将介绍在 JavaScript 中处理事件的方法及 XHTML 元素事件的种类。

1. 指定事件处理程序

指定事件处理程序有以下三种方法。

(1) 直接在 XHTML 元素中指定

这种方法应用比较普遍，具体格式如下：

```
<XHTML 元素...事件="事件处理程序"[事件="事件处理程序"...]>
```

例如，在<body>元素中指定 onload 事件的代码如下：

```
<body onload="alert('网页读取完成，请欣赏！')" onunload="alert('欢迎下次光临，再见！')">
```

网页加载完毕后运行结果如图 4.2 所示，网页关闭后运行结果如图 4.3 所示。

在上述代码中，<body>元素能使页面读取完毕时弹出一个对话框，并显示提示信息“网页读

取完成, 请欣赏!"; 在用户退出页面 (此处的“用户退出页面”实际上指的是用户刷新页面的操作, 而非关闭整个浏览器窗口) 时也弹出一个对话框, 并显示提示信息“欢迎下次光临, 再见!”。

(2) 编写特定对象特定事件的 JavaScript

这种方法应用比较少, 但在某些实际操作过程中比较有用, 具体格式如下:

```
<script language = "JavaScript" for = "对象" event = "事件">
    ...(事件处理程序代码)...
</script>
```

例如, 弹出一个提示对话框的 JavaScript 代码如下:

```
<script language = "JavaScript" for = "window" event = "onload">
    alert('网页读取完成, 请慢慢欣赏! ');
</script>
```

程序运行结果如图 4.4 所示。



图 4.2 网页加载完毕后运行结果



图 4.3 网页关闭后运行结果

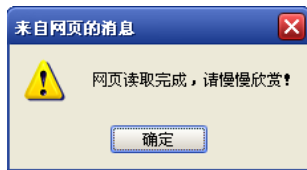


图 4.4 弹出一个提示对话框

(3) 在 JavaScript 中说明

具体格式如下:

```
<事件主角-对象>.<事件> = <事件处理程序>;
```

在上述格式中, “事件处理程序” 是真正的代码, 而不是字符串形式的代码。如果事件处理程序是自定义函数, 没有任何参数, 就不要加 “()”。例如下面的代码:

```
function ignoreError()
{
    return true;
}
...
window.onerror = ignoreError; //没有使用"()"
```

在上述代码中, ignoreError() 函数定义 window 对象的 onerror 事件的处理程序。它的功能是忽略该 window 对象下的任何错误, 但引用不允许访问的 location 对象产生的“没有权限”错误是不能忽略的。

2. 鼠标单击事件

鼠标单击事件是常见的事件, 事件对应的方法名是 onclick, 具体格式如下:

```
onclick = 函数或处理语句
```

例如, 单击网页中的按钮后, 弹出提示信息的代码如下:

```
<body>
    <input type = "button" value = "单击按钮" onclick = "alert('鼠标单击事件')">
</body>
```

程序运行结果如图 4.5 所示。

3. 下拉列表事件

下拉列表是常用的一种 XHTML 元素, 通常情况下, 利用 onChange 事件来处理, 具体格式如下:

onChange = 函数或处理语句



图 4.5 鼠标单击事件的运行结果

例如，选中网页中的下拉列表内容，弹出提示信息的代码如下：

```
<select name="select" onchange="alert('您选择了'+select.value)">
  <option value="北京">北京</option>
  <option value="上海">上海</option>
  <option value="天津">天津</option>
  <option value="重庆">重庆</option>
  <option value="南京">南京</option>
</select>
```

程序运行结果如图 4.6 所示。



图 4.6 选中网页中的下拉列表内容的运行结果

4. 判断输入框是否为空

当用户浏览网页时，经常需要进行“注册”或“登录”的操作，需要校验表单中输入框是否为空。这时，可以利用<form>元素中的 onsubmit 属性进行设置，该事件发生在表单的“提交”按钮被单击（按下并放开）时，用于验证表单的有效性。可以通过在事件处理程序中返回 false 值（return false）阻止表单提交。

【例 4.2】校验用户注册页面中的表单内容是否为空。

编写用户注册页面中的表单信息代码如下：

```
<html>
<body>
  <meta http-equiv="content-type" content="text/html; charset=gb2312">
```

```

<form action="" method="post" name="form1" onsubmit="return userCheck()">
  <table width="409" border="1">
    <tr>
      <td>用户名: </td>          <!--设置用户名的表单-->
      <td><input type="text" name="username"></td>
    </tr>
    <tr>
      <td>密码: </td>          <!--设置用户登录密码的表单-->
      <td><input type="password" name="password"></td>
    </tr>
    <tr>
      <td>确认: </td>          <!--设置确认密码的表单-->
      <td><input type="password" name="repassword"></td>
    </tr>
    <tr>
      <td>出生: </td>          <!--设置出生日期的表单-->
      <td><input type="text" name="born"></td>
    </tr>
    <tr>
      <td>地址: </td>          <!--设置地址的表单-->
      <td><input type="text" name="address"></td>
    </tr>
    <tr>
      <td>介绍: </td>          <!--设置介绍的表单-->
      <td><textarea name="introduce" rows="5" id="introduce"></textarea></td>
    </tr>
  </table>
  <input type="submit" name="Submit1" value="注册">
  <input type="reset" name="Submit2" value="重置">
</form>
</body>
</html>

```

校验用户注册页面的表单信息是否为空的 JavaScript 代码如下:

```

<script type="text/javascript">
  function userCheck(){
    //校验用户名表单是否为空
    if (document.form1.username.value=="") {
      window.alert("请输入用户名");
      return false;
    }
    //校验密码表单是否为空
    if (document.form1.password.value=="") {
      window.alert("请输入用户密码");
      return false;
    }
    //校验密码确认是否为空

```

```
if (document.form1.repassword.value=="") {  
    window.alert("请输入密码确认");  
    return false;  
}  
//校验密码与确认密码是否一致  
if (document.form1.repassword.value!=document.form1.password.value) {  
    window.alert("您输入的两次密码并不相同");  
    return false;  
}  
//校验出生日期表单是否为空  
if (document.form1.born.value=="") {  
    window.alert("请输入出生日期");  
    return false;  
}  
//校验地址表单是否为空  
if (document.form1.address.value=="") {  
    window.alert("请输入地址");  
    return false;  
}  
//校验自我介绍表单是否为空  
if (document.form1.introduce.value=="") {  
    window.alert("请输入自我介绍");  
    return false;  
}  
return true;  
}  
</script>
```

将以上两段代码编辑在同一个记事本文件中，用文件名 4_2usrchk.html 保存。程序运行结果如图 4.7 所示。



图 4.7 验证表单内容是否为空

4.2 JavaScript 浏览器对象

4.2.1 浏览器对象的概念

为了方便网页脚本语言的编程，早在 IE3.0 与 Netscape Navigator 3.0 时代，主流浏览器厂商就在自己的产品中实现并公开了一些可在脚本中访问和使用的对象，网页编程人员在自己编写的代码中直接使用这些对象，来访问和操控浏览器窗口，比如移动窗口、更改状态栏文本以及执行其他不与页面内容发生直接联系的操作。

随着浏览器的升级换代，这类浏览器自身内置的对象种类越来越多，功能也日趋强大，并逐步形成了一套通用的体系结构，称为**浏览器对象模型（Browser Object Model, BOM）**，这些对象也就统称为**浏览器对象**。之前我们在编程中用到的 document、window 等就是最常用的浏览器对象。

浏览器的对象具有树形结构，如图 4.8 所示。

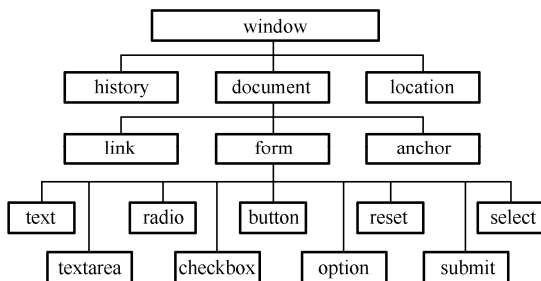


图 4.8 浏览器对象树

使用浏览器的内部对象系统，可实现与 HTML/XHTML 文档进行交互。它将相关元素组织包装起来，提供给程序开发人员，从而减轻编程人员的负担，提高设计 Web 页面的能力。它的作用主要体现在以下三个方面。

（1）编程人员可利用这些对象，对浏览器环境中的事件进行控制并做出处理。

（2）JavaScript 中提供了非常丰富的内部方法和属性，从而减轻了编程人员的负担，提高了开发效率。

（3）在对象系统中，文档对象是非常重要的，它位于底层，对我们实现 Web 页面信息交互起关键作用。

接下来仅就图 4.8 中的几个主要的常用对象分别加以介绍。

4.2.2 Window 对象

Window 对象描述浏览器窗口特征，它是 Document、Location 和 History 对象的父对象。另外，还可以认为它是其他任何对象的假定父对象。

例如：

```
alert("欢迎 2014 南京青奥会");
```

相当于如下语句:

```
Window.alert("喜迎 2014 南京青奥会");
```

1. Window 对象属性

- **name**: 指定窗口的名称。浏览器可同时打开多个窗口, 利用窗口名称可以区分它们。用 Window 对象的 **open** 方法打开一个新窗口时可指定窗口名称; a 标记的 **target** 属性指定窗口的名称, 单击该锚点可链接到该窗口。下例中的超链接将打开一个 **name** 属性为 “IE_Window” 的 Window 对象。

```
<a href="http://www.njnu.edu.cn" target="IE_Window">南京师范大学</a>
```

- **parent**: 代表当前窗口 (框架) 的父窗口, 使用它返回对象的方法和属性。
- **opener**: 返回产生当前窗口的窗口对象, 使用它返回对象的方法和属性。
- **top**: 代表主窗口, 是顶层的窗口, 也是所有其他窗口的父窗口。可通过该对象访问当前窗口的方法和属性。
- **self**: 返回当前窗口的一个对象, 可通过该对象访问当前窗口的方法和属性。
- **defaultstatus**: 返回或设置将在浏览器状态栏中显示的默认内容。
- **status**: 返回或设置将在浏览器状态栏中显示的指定内容。例如, 在浏览器状态栏中显示浏览当天的日期:

```
status = Dateformat(date);
```

2. Window 对象的方法

- **alert()**: 显示一个警告对话框, 包含一条信息和一个 “确定” 按钮。

语法格式如下:

```
alert(参数)
```

它的参数就是提示信息。执行 **alert()** 方法时, 脚本的执行过程会暂停下来, 直到用户单击 “确定” 按钮。例如:

```
Window.alert("欢迎访问南京师范大学");
```

- **confirm()**: 显示一个确认对话框, 包含一条指定信息, 还包含 “确定” 按钮和 “取消” 按钮。

语法格式如下:

```
confirm(参数)
```

它的参数就是提示信息。单击 “确定” 按钮, 返回 **true**; 单击 “取消” 按钮, 则返回 **false**。例如下面的语句:

```
Res=confirm("欢迎访问南京师范大学")
```

```
if Res then Form.Submit
```

- **prompt()**: 显示一个提示对话框, 提示用户输入数据。

语法格式如下:

```
prompt(参数 1, 参数 2)
```

参数 1 给出提示信息, 参数 2 指定默认响应。执行 **prompt()** 方法时, 将显示一个提示对话框, 让用户在文本框中输入字符串, 完成输入后, 单击 “确定” 按钮, 返回所输入的字符串; 单击 “取消” 按钮, 则不返回任何信息。其作用类似于 **inputBox** 函数。

- **open()**: 打开一个已存在的窗口, 或者创建一个新窗口, 并在该窗口中加载一个文档。

语法格式如下:

```
NewWindow = Window.open(url, name, 窗口参数设置表)
```

其中, `NewWindow` 用于接收 `open()` 方法的返回值, 它是一个 `Window` 对象。参数 `url` 指定要在窗口中显示的文档的 URL; 参数 `name` 指定要打开的窗口名称。如果指定窗口已存在, 则在该窗口显示新文档, 原有内容被取代; 如果指定窗口不存在, 则以指定名称创建并打开一个新窗口, 并且在该窗口中显示新文档内容。

窗口参数设置表格式:

参数 1=值, 参数 2=值, ...

窗口参数用于描述打开的窗口, 参数可以有多个, 是可选项。例如:

```
Set NewWindow1=Window.open("Jsp.htm","WindowIE","toolbar=no,location=no")
```

这行语句将在 `WindowIE` 窗口打开 `Jsp.htm` 文件, 并且产生一个句柄为 `NewWindow1` 的对象。

- `close()`: 关闭一个打开的窗门。例如, 在 `Mywin` 窗口中打开 `example.htm` 页面, 该窗口没有状态栏、工具栏、菜单栏和地址栏。

```
Mywin=Window.open("example.htm","mywin","Status=no, toolbar=no, menubar=no, location=no");
```

关闭这个打开的窗口, 语句如下:

```
Mywin.close()
```

- `navigate()`: 在当前窗口中显示指定网页。

语法格式如下:

```
navigate (url)
```

其中 `url` 参数用于指定要显示的新文档的 URL。例如, 在当前窗口打开南京师范大学主页:

```
Window.navigate ("http://www.njnu.edu.cn");
```

- `setTimeout()`: 设置一个计时器, 在经过指定的时间间隔后调用一个过程。

语法格式如下:

```
变量名=Window.setTimeout(过程名,时间间隔,脚本语言)
```

其中, 变量名保存 `setTimeout()` 方法的返回值, 它是一个 `Timer` 对象。过程名给出到指定的时间间隔要调用的过程或函数的名称。时间间隔以毫秒为单位。例如, 打开窗口 3s 后调用 `MyProc` 过程:

```
TID=Window.setTimeout("MyProc", 3000, "JavaScript");
```

- `clearTimeout()`: 给指定的计时器复位。

语法格式如下:

```
Window.clearTimeout(对象)
```

其中, “对象”是用 `setTimeout()` 方法返回的计时器对象。例如:

```
Window.clearTimeout (TID)
```

这行代码可以清除名字为 “TID” 的计时器对象。

- `focus()`: 使一个 `Window` 对象得到焦点。例如, 要使 `NewWindow` 对象得到焦点, 使用如下语句:

```
NewWindow.focus();
```

- `blur()`: 使一个 `Window` 对象失去焦点。例如, 要使 `NewWindow` 对象失去焦点, 使用如下语句:

```
NewWindow.blur();
```

3. Window 对象的事件

`Window` 对象事件如表 4.8 所示。

表 4.8 Window 对象事件

事 件	说 明
OnLoad	HTML 文件载入浏览器时发生
OnUnLoad	HTML 文件从浏览器删除时发生
OnFocus	窗口获得焦点时发生
OnBlur	窗口失去焦点时发生
OnHelp	用户按下 F1 键时发生
OnResize	用户调整窗口大小时发生
OnScroll	用户滚动窗口时发生
OnError	载入 HTML 文件出错时发生

4.2.3 Document 对象

Document 对象表示在浏览器窗口或其中一个框架中显示的 HTML/XHTML 文档，通过该对象的属性和方法可以获得和控制文档的外观和内容。

Document 对象包含以下对象和集合：All（文档中所有元素的集合）、Anchors（锚点集合）、Applets（Java 小程序集合）、Body（文档主体对象）、Children（子元素集合）、Embeds（嵌入对象）、Forms（表单集合）、Frames（框架集合）、Images（图像集合）、Links（链接集合）、Plugins（插件集合）、Scripts（脚本集合）、Selection（选择器对象）和 StyleSheets（级联样式表集合）。通过这些集合可以获取网页中某一类型的所有元素，并可通过索引来访问集合中的指定元素。

1. Document 对象的属性

Document 对象有许多属性，用来设置文档的背景颜色、链接颜色和文档标题等，也可执行更为复杂的操作。

（1）与颜色有关的属性

- fgColor：设置或返回文档的文本颜色。
- bgColor：设置或返回文档的背景颜色。它与 body 标记的 bgcolor 属性功能相同。
- linkColor：设置或返回文档中超链接的颜色。它与 body 标记的 link 属性功能相同。使用方法如下：

```
Window.document.linkColor = color;
```

其中 color 是一种颜色的描述。它是颜色名称或颜色的数值表示。例如，颜色的名称可以是 green，颜色的数值可以是#C00000。

linkColor 的值在网页首次载入时设置，随后可以重新设置和修改。

- aLinkColor：设置或返回文档中活动链接的颜色。活动链接是鼠标指针指向一个超链接，按下鼠标左键但尚未释放时的状态。它与 body 标记的 aLink 属性功能相同。
- vLinkColor：设置或返回已经访问过的超链接的颜色，与 body 标记中的 vLink 属性功能相同。

（2）与 HTML 文件有关的属性

- title：返回当前文档的标题，在运行期间不能改变。
- location：设置或返回文档的 URL。
- parentWindow：包含此 HTML 文件的上层窗口。

- **referrer**: 返回链接到当前页面的那个页面的 URL。
- **lastModified**: 返回当前文档的最后修改日期。

(3) 对象属性

对象属性就是对象属性的值。例如, 通过 **length** 属性可以返回当前文档中该对象的数目。每个对象都被存储在数组中, 可以通过索引值来访问该数组中的元素。

- **all**: 返回所有标记和对象。
- **anchors**: 表示文档中的锚点, 每个锚点都被存储在 **anchors** 数组中。
- **links**: 表示文档中的超链接, 每个超链接都被存储在 **links** 数组中。
- **forms**: 返回所有表单。
- **images**: 返回所有图像。
- **styleSheets**: 返回所有样式属性对象。
- **applets**: 返回所有 Applet 对象。
- **embeds**: 返回所有嵌入标记。
- **scripts**: 返回所有 Script 程序对象。

2. Document 对象的方法

Document 对象通过方法对文档内容进行控制。

- **open()**: 打开要输入的文档。执行该方法后, 文档中的当前内容被清除, 可以使用 **write** 或 **writeln** 方法将新内容写到文档中。

语法格式:

```
Document.open(参数 1,参数 2);
```

- **write()**: 向文档中写入 HTML 代码。

语法格式:

```
Document.write(写入内容);
```

执行 **write** 方法后, 写入内容插入文档的当前位置, 但该文档要执行 **close()** 方法后才能显示出来。

- **writeln()**: 向文档中写入 HTML 代码。

语法格式:

```
Document.writeln(写入内容);
```

writeln() 方法与 **write()** 方法类似, 不同的是 **writeln()** 在内容末尾添加一个换行符。

- **close()**: 关闭文档, 并显示所有使用 **write()** 或 **writeln()** 方法写入的内容。
- **clear()**: 清除当前文档的内容, 刷新屏幕。

对于 Document 对象的各个方法, 浏览器默认的在当前文档中放入数据时使用的各种方法的顺序通常如下所示:

```
Document.Open();  
Document.Write(content);  
Document.Close();
```

其中 **content** 可以是一个字符串或一个有确定值的变量。

3. Document 对象的事件

Document 对象的事件主要有鼠标事件和键盘事件, 如表 4.9 所示。

表 4.9 Document 对象事件

事 件	说 明
onClick	单击鼠标
onDbClick	双击鼠标
onMouseDown	按下鼠标左键
onMouseUp	放开鼠标左键
onMouseOver	鼠标移到对象上
onMouseOut	鼠标离开对象
onMouseMove	移动鼠标
onSelectStart	开始选取对象内容
onDragStart	开始以拖动方式移动选取对象内容
onKeyDown	按下键盘按键
onKeyPress	用户按下任意键时，先产生 KeyDown 事件。若用户一直按住按键，则产生连续的 KeyPress 事件

4.2.4 History 对象

History 对象包含用户已经浏览过的 URL 集合，提供浏览器导航按钮功能，可以通过文档的历史记录来浏览文档。

- 1. History 对象的属性
 - length: 返回历史表中的 URL 地址数目。
- 2. History 对象的方法
 - back(): 在历史表中向后搜索。
 - forward(): 在历史表中向前搜索。
 - go(): 在历史表中跳转到指定的项。

4.2.5 Navigator 对象

Navigator 对象包含浏览器的信息。

- 1. Navigator 对象的属性
 - appCodeName: 返回浏览器的代码名称。对于 IE 浏览器，返回 Mozilla。
 - appName: 返回浏览器的名称。对于 IE 浏览器，返回 Microsoft Internet Explorer。
 - appVersion: 返回浏览器的版本号。
 - userAgent: 返回当前用户所使用的语言。如果用户使用简体中文 Windows，则返回 zh-cn。
 - cookieEnabled: 如果允许使用 cookies，则该属性返回 true，否则返回 false。
- 2. Navigator 对象的方法

它提供了一种用于确定浏览器中的 Java 是否可用的方法：

```
javaEnabled();
```

如果 Java 可用，返回值为 true，否则为 false。

4.2.6 Location 对象

Location 对象包含当前 URL 的信息。

1. Location 对象的属性

- href: 返回或设置当前文档的完整 URL。
- hash: 返回或设置当前 URL 中#后面的部分（即书签）的名称。
- host: 返回或设置当前 URL 中的主机名和端口部分。
- hostname: 返回或设置当前 URL 中的主机名。
- port: 返回或设置当前 URL 中的端口部分。
- path: 返回或设置当前 URL 中的路径部分。
- protocol: 返回或设置当前 URL 中的协议类型。
- search: 返回或设置当前 URL 中的查询字符串，即提交给服务器时在 URL 中紧跟在问号后面的内容。如果当前 URL 中不包含查询字符串，则它返回一个空字符串。

2. Location 对象的方法

- reload(): 重新加载当前文档。
- replace(): 用参数中给出的网址替换当前的网址。
- assign(): 加载 URL 指定的新的 HTML 文档。

4.2.7 Link 对象

Link 对象表示文档中的超链接，通过该对象的一些属性可以得到链接目标。Link 对象的基本属性是 length，它返回文档中链接的数目。每个链接都是 Links 数组中的一个元素，可以通过索引值来访问。例如，第一个链接是 Links(0)，第二个链接是 Links(1)，最后一个链接是 Links(Links.Length-1)。

Link 对象的大多数属性与 Location 对象的属性基本相同，不再赘述。

4.3 JavaScript 页面开发实例

本节通过 3 个综合实例，来巩固前两节所学的有关 JavaScript 编程及浏览器对象应用的知识。

4.3.1 制作隐式菜单

【例 4.3】用 JavaScript 制作隐式菜单，即菜单显示时，先将子菜单隐藏起来，只显示主菜单。当鼠标移至主菜单上时，才显示子菜单；当鼠标离开菜单区域后，子菜单又隐藏起来。

输入以下内容，以 4_3hideMenu.html 作为文件名保存：

```
<html>
<head>
  <title>隐式菜单精彩实例</title>
</head>
<script language="javascript">
```

```

function move(x) {
    if (document.all) {
        object1.style.pixelLeft += x;
        object1.style.visibility = "visible";
    }
    else if (document.layers) {
        document.object1.left += x;
        document.object1.visibility = "show";
    }
}
</script>
<body leftmargin="20" bgcolor="#ffcccc">
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
    <layer visibility="hide" top="20" name="object1" bgcolor="black" left="0"
        onmouseover="move(132)" onmouseout="move(-132)">
    <script language="javascript">
        function positionmenu(){
            move(-132);
        }
        if (document.all) {
            document.write('<div id="object1" style="visibility:hidden;cursor:hand;Position:Absolute;
            Left:0px;Top:20px;Z-Index:20" onmouseover="move(132)" onmouseout="move(-132)">');
        }
    </script>
    <table border="0" cellpadding="0" cellspacing="1" width="150" bgcolor="#00f0f0">
        <tr>
            <td bgcolor="#0099FF" align="center">
                <span style="font-size:11pt;font-family:宋体"><b>院校导航</b></span>
            </td>
            <td align="center" rowspan="100" width="20" bgcolor="#ff8888">
                <span style="font-size:13px;font-family:宋体">
                    <p align="center"><b>南<br>京<br>高<br>校<br>网<br>站<br>入<br>口</b></p>
                </span>
            </td>
        </tr>
        <tr>
            <td>
                <a href="http://www.nju.edu.cn/">南京大学</a>
            </td>
        </tr>
        <tr>
            <td>
                <a href="http://www.seu.edu.cn/">东南大学</a>
            </td>
        </tr>
        <tr>
            <td>
                <a href="http://www.njnu.edu.cn/">南京师范大学</a>
            </td>
        </tr>
    </table>

```

```

        <td>
            <a href="http://www.njut.edu.cn/">南京工业大学</a>
        </td>
    </tr>
    <tr>
        <td>
            <a href="http://www.njfu.edu.cn/">南京林业大学</a>
        </td>
    </tr>
    <tr>
        <td>
            <a href="http://www.njupt.edu.cn/">南京邮电大学</a>
        </td>
    </tr>
    <tr>
        <td bgcolor="#0099ff"></td>
    </tr>
</table>
<script language="javascript">
    if (document.all) {
        document.write('</div>');
    }
    window.onload=positionmenu;
</script>
</layer>
</body>
</html>

```

在以上代码中，编辑 `move()` 函数来控制菜单的隐藏及可见；鼠标事件为 `onmouseover` 时触发显示，为 `onmouseout` 时隐藏。

运行 `4_3hideMenu.html` 文件，显示如图 4.9 所示的页面。

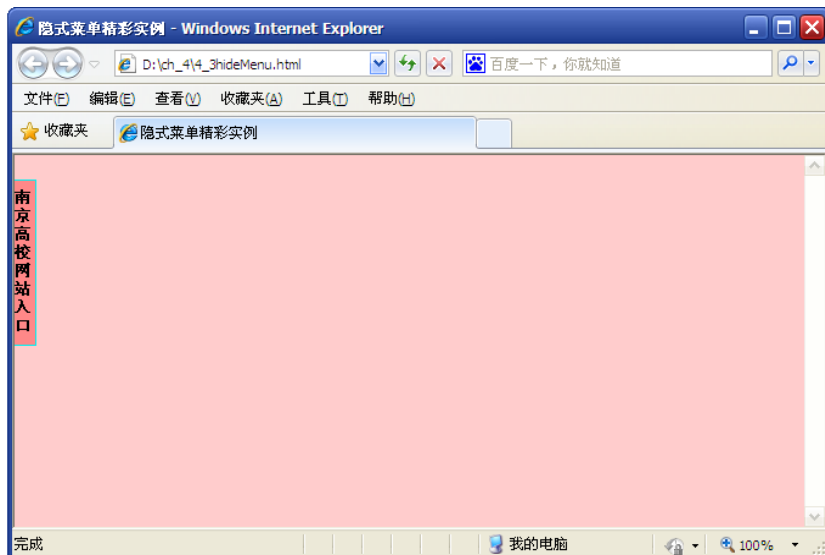


图 4.9 运行隐式菜单

当将鼠标移至左侧“南京高校网站入口”处时，将显示如图 4.10（上）所示的画面。此时单击“南京师范大学”，即可打开相应链接，转到南京师范大学网站的主页，如图 4.10（下）所示。



图 4.10 显示子菜单、打开链接页

4.3.2 青奥会倒计时牌

【例 4.4】用 JavaScript 实现一个倒计时显示牌，动态显示距离 2014 年南京青奥会开幕还有多长时间（精确到秒）。

输入以下内容，以 4_4yogdna.html 作为文件名保存：

```
<html>
<head>
  <title>倒计时显示牌</title>
</head>
  <script language="javascript">
    <!-- hide script from old browser
      var LeaveHour = -1
      var LeaveMinute = -1
      var LeaveSecond = -1
```

```

//初始化时间, 用以做比较
var Tday = new Date("8/15,2014 23:59:59")
var daysms = 24 * 60 * 60 * 1000
var hoursms = 60 * 60 * 1000
var Secondms = 60 * 1000
var microsecond = 1000
function clock(){
    //取出当前系统的时间, 以及小时、分钟、秒
    var nowtime = new Date()
    var hour = nowtime.getHours()
    var minute = nowtime.getMinutes()
    var second = nowtime.getSeconds()
    //将二十四进制时间格式转为十二进制
    var timevalue = "" + ((hour > 12) ? hour-12:hour)
    timevalue += ((minute < 10) ? ":0":":") + minute
    timevalue += ((second < 10) ? ":0":":") + second
    timevalue += ((hour > 12) ? " PM":" AM")
    var convertHour = LeaveHour
    var convertMinute = LeaveMinute
    var convertSecond = LeaveSecond
    //取时间差, 用 Math 对象的 floor 方法对时间差取整
    var Diffms = Tday.getTime() - nowtime.getTime()
    LeaveHour = Math.floor(Diffms / daysms)
    Diffms -= LeaveHour * daysms
    LeaveMinute = Math.floor(Diffms / hoursms)
    Diffms -= LeaveMinute * hoursms
    LeaveSecond = Math.floor(Diffms / Secondms)
    Diffms -= LeaveSecond * Secondms
    var dSecs = Math.floor(Diffms / microsecond)
    if(convertHour != LeaveHour)
        //在相应的文本框上显示时间
        document.formnow.dd.value=LeaveHour
    if(convertMinute != LeaveMinute)
        document.formnow.hh.value=LeaveMinute
    if(convertSecond != LeaveSecond)
        document.formnow.mm.value=LeaveSecond
    document.formnow.ss.value=dSecs
    setTimeout("clock()",1000)
}
-->
</script>
<body onload="clock();return true">
<meta http-equiv="content-type" content="text/html; charset=gb2312">
<form name="formnow">
    <font color="blue" size="5">现在距离 2014 年南京青奥会开幕还有:</font><br><br>
    <input type="text" name="dd" size="2"> 天
    <input type="text" name="hh" size="2"> 小时
    <input type="text" name="mm" size="2"> 分

```

```
<input type="text" name="ss" size="2"> 秒  
</form>  
</body>  
</html>
```

运行效果如图 4.11 所示。



图 4.11 青奥会倒计时牌的运行效果

本例说明。

① 在<body>区域设置 onload 属性为 “clock();return true”，触发 clock()函数。建立表单，设置 name 属性为 “formnow”，添加 4 个文本框用来显示时间。

② setTimeout()函数设定每 1000 毫秒调用一次 clock ()，刷新一次信息，以达到动态显示的效果。

4.3.3 图像自由运动

【例 4.5】利用 JavaScript 实现图像在浏览器窗口的自由运动，当越出边界时，从任意位置返回窗口继续自由运动。

(1) 准备

从网上下载一个图像文件作为本例演示所用的资源（这里选取的是一条热带观赏鱼的照片），命名为 fish.jpg，与源文件 4_5freeMove.html 一同放置在 D:\ch_4 目录下，如图 4.12 所示。

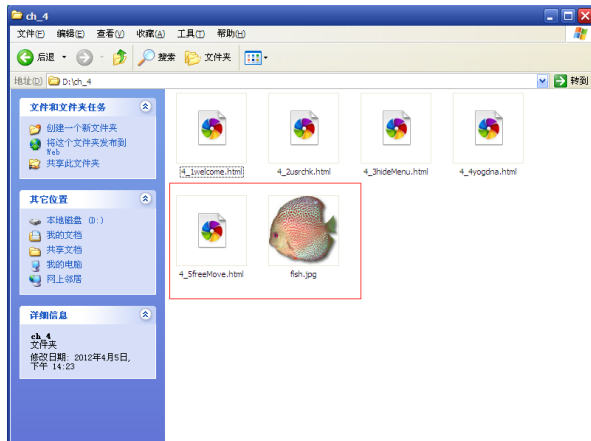


图 4.12 准备图片资源

(2) 编写 html 程序

编辑 4_5freeMove.html 文件，实现图像的自由运动，实现代码如下：

```
<html>
<head>
    <title>图像自由运动</title>
</head>
<body>
    <meta http-equiv="content-type" content="text/html; charset=gb2312">
    <script language="javascript">
        <!--
            var no = 5, speed = 40;                                //①
            var heart = "fish.jpg";
            var flag;
            var ns4up = (document.layers) ? 1 : 0;
            var ie4up = (document.all) ? 1 : 0;
            var dx, xp, yp, am, stx, sty;
            var i, doc_width = 600, doc_height = 400;
            if (ns4up) {
                doc_width = self.innerWidth;
                doc_height = self.innerHeight;
            }
            else if (ie4up) {
                doc_width = document.body.clientWidth;
                doc_height = document.body.clientHeight;
            }
            dx = new Array();
            xp = new Array();
            yp = new Array();
            amx = new Array();
            amy = new Array();
            stx = new Array();
            sty = new Array();
            flag = new Array();                                    //②
            for (i = 0; i < no; ++ i) {                            //no 为生成图片的张数，本例是 5 张
                dx[i] = 0;
                xp[i] = Math.random()*(doc_width-30)+10;          //随机产生图片位置的坐标
                yp[i] = Math.random()*doc_height;
                amy[i] = 12+ Math.random()*20;
                amx[i] = 10+ Math.random()*40;
                stx[i] = 0.02 + Math.random()/10;
                sty[i] = 0.7 + Math.random();
                flag[i] = (Math.random()>0.5)?1:0;                //③
                if (ns4up) {                                        //设置不同浏览器
                    if (i == 0) {
                        document.write("<layer name=\"dot\"+ i +\"\" left=\"15\" ");
                        document.write("top=\"15\" visibility=\"show\"><img src=\"\"");
                        document.write(heart+ "\" border=\"0\"></layer>");
                    } else {
```

```

        document.write("<layer name=\"dot"+ i +\"" left=\"15\" ");
        document.write("top=\"15\" visibility=\"show\"><img src=\"");
        document.write(heart+ "\" border=\"0\"></layer>");
    }
} else if (ie4up) { //④
    if (i == 0) {
        document.write("<div id=\"dot"+ i +\"" style=\"position: ");
        document.write("absolute; z-index: "+ i +"; visibility: ");
        document.write("visible; top: 15px; left: 15px;\"><img src=\"");
        document.write(heart+ "\" border=\"0\"></div>");
    } else {
        document.write("<div id=\"dot"+ i +\"" style=\"position: ");
        document.write("absolute; z-index: "+ i +"; visibility: ");
        document.write("visible; top: 15px; left: 15px;\"><img src=\"");
        document.write(heart+ "\" border=\"0\"></div>");
    }
}
}
function move() {
    for (i = 0; i < no; ++ i) {
        if (yp[i] > doc_height-50) {
            xp[i] = 10 + Math.random()*(doc_width-amx[i]-30);
            yp[i] = 0;
            flag[i]=(Math.random()<0.5)?1:0;
            stx[i] = 0.02 + Math.random()/10;
            sty[i] = 0.7 + Math.random();
            doc_width = self.innerWidth;
            doc_height = self.innerHeight;
        }
        if (flag[i])
            dx[i] += stx[i];
        else
            dx[i] -= stx[i];
        if (Math.abs(dx[i]) > Math.PI) {
            yp[i]+=Math.abs(amy[i]*dx[i]);
            xp[i]+=amx[i]*dx[i];
            dx[i]=0;
            flag[i]=!flag[i];
        }
        document.layers["dot"+i].top = yp[i] + amy[i]*(Math.abs(Math.sin(dx[i])+dx[i]));
        document.layers["dot"+i].left = xp[i] + amx[i]*dx[i];
    }
    setTimeout("move()", speed); //⑤
}
function to_move() {
    for (i = 0; i < no; ++ i) {
        if (yp[i] > doc_height-50) {
            xp[i] = 10+ Math.random()*(doc_width-amx[i]-30);

```

```

        yp[i] = 0;
        stx[i] = 0.02 + Math.random()/10;
        sty[i] = 0.7 + Math.random();
        flag[i]=(Math.random()<0.5)?1:0;
        doc_width = document.body.clientWidth;
        doc_height = document.body.clientHeight;
    }
    if (flag[i])
        dx[i] += stx[i];
    else
        dx[i] -= stx[i];
    if (Math.abs(dx[i]) > Math.PI) {
        yp[i]+=Math.abs(amy[i]*dx[i]);
        xp[i]+=amx[i]*dx[i];
        dx[i]=0;
        flag[i]=!flag[i];
    }
    document.all["dot"+i].style.pixelTop = yp[i]
        + amy[i]*(Math.abs(Math.sin(dx[i])+dx[i]));
    document.all["dot"+i].style.pixelLeft = xp[i] + amx[i]*dx[i];
}
    setTimeout("to_move()", speed);           //⑤
}
if (ns4up) {
    move();                                   //⑥
}
else if (ie4up) {
    to_move();                               //⑦
}
//-->
</script>
</body>
</html>

```

双击运行该程序，就能在浏览器中看到 5 条美丽的鱼儿翩翩起舞的神奇特效，如图 4.13 所示。

（3）代码分析

① 首先声明变量，赋初值。指定生成 5 张图片，即 `no=5`，并且设定每 40 毫秒刷新一次，即 `speed=40`。

② 声明了一组数组，用于存放图片运动的坐标值。

③ 使用 `Math` 对象的 `random()` 方法产生随机数，参与生成每张图片的初始位置，从而实现真正的自由轨迹。

④ 针对不同的浏览器，分别实现图片运动的方法。

⑤ 设置图片运动的速度。

⑥ 在 Netscape 浏览器中由 `move()` 方法确定图片的运行轨迹。

⑦ 在 IE 浏览器中由 `to_move()` 方法确定图片的运行轨迹。

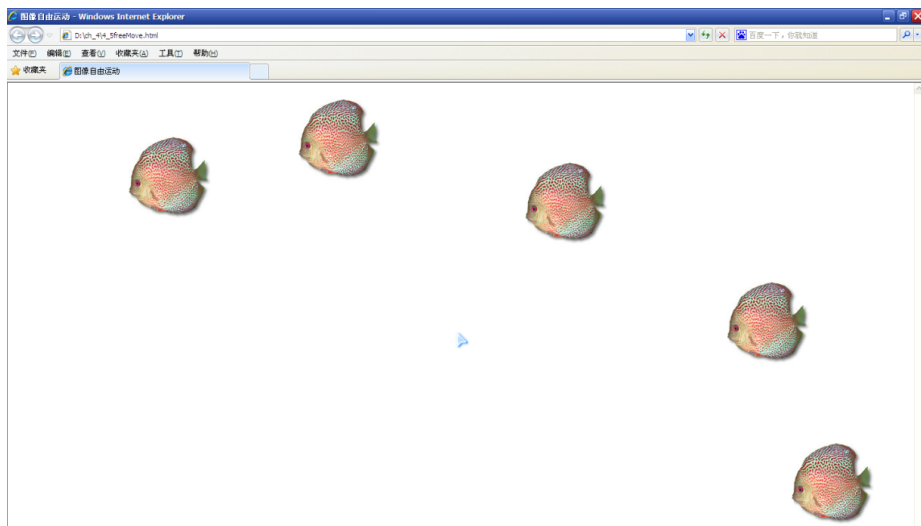


图 4.13 鱼儿们自由游动的特效

4.4 Ajax 技术

除了 JavaScript，当前 JSP 前端开发中最流行的技术还有 Ajax。

4.4.1 Ajax 的概念

Ajax 是异步 JavaScript 和 XML (Asynchronous JavaScript and xml) 的英文缩写，这个名词是由 Jesse James Garrett 首先提出的，而大力推广且使 Ajax 技术炙手可热的是 Google 公司。Google 公司发布的 Gmail、Google Suggest 等应用最终让人们体验到了什么是 Ajax。

Ajax 的核心理念是使用 XMLHttpRequest 对象发送异步请求。最初为 XMLHttpRequest 对象提供浏览器支持的是微软。早在 1998 年微软公司开发 Web 版 Outlook 时，就已经以 ActiveX 控件的方式为 XMLHttpRequest 提供了支持。

事实上，Ajax 并不是一种全新的技术，而是几种技术的融合。

- HTML/XHTML：用于页面内容的表示。
- CSS：格式化文本内容。
- DOM：对页面内容进行动态更新。
- XML：实现数据交换和格式转化。
- XMLHttpRequest 对象：与服务器异步通信。

而 JavaScript 则实现以上所有技术的融合，以上每种技术都具有独特之处，融合在一起就形成了一种功能强大的新技术。

现在，许多应用程序都是在 Web 上创建的。但是，Web 也成为限制 Web 应用程序发展的瓶颈，这源于网络延迟的不确定性。网络连接是耗费资源的行为，程序必须序列化，通信协议沟通及路由传输等动作都很浪费时间和资源。在 Web 应用程序中，数据通常通过表单提交，在同步情况下，使用者发送表单之后，就只能等待服务器回应，而在这段时间内无法进行进一步操作，如图 4.14 所示。

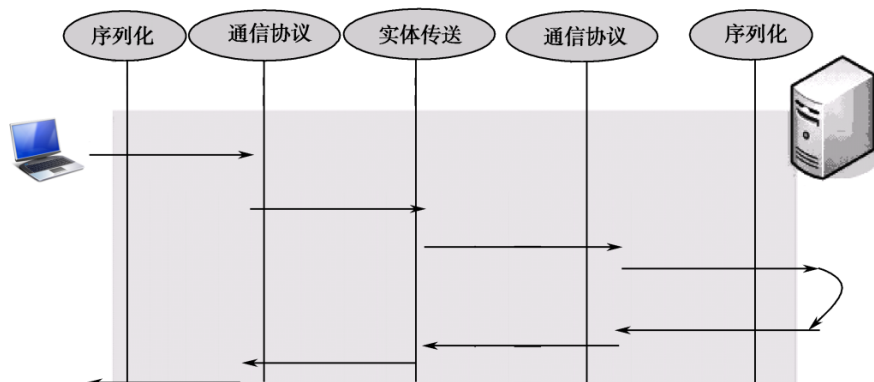


图 4.14 同步情况

图 4.14 中加阴影的部分是发送表单之后使用者必须等待的时间，浏览器预设为使用同步方式送出请求并等待回应。

如果把请求与回应改为非同步进行，即发出请求后，浏览器不需要苦等服务器的回应，而让使用者对浏览器中的 Web 程序进行其他操作，当服务器处理完请求并送出回应时，计算机接收到回应，再呼叫浏览器所设定的对应动作进行处理，如图 4.15 所示。

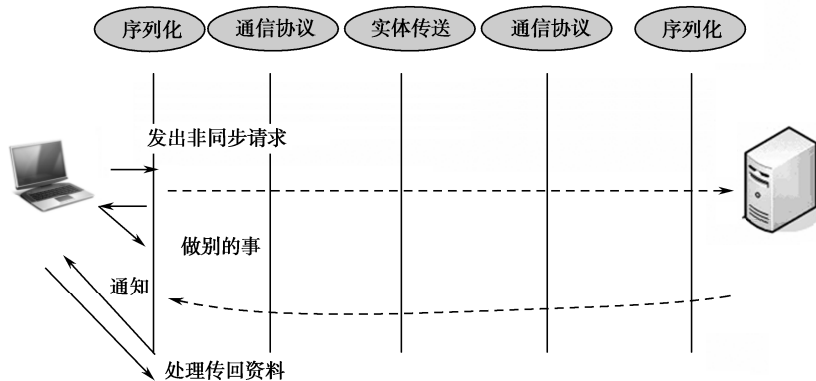


图 4.15 非同步情况

现在的问题是，由谁来发这个非同步请求？它正是 Ajax 的 XMLHttpRequest 组件！这个组件通过 JavaScript 建立，在不同的浏览器产品中表现为不同的形态，在 Firefox、NetScape、Safari、Opera 中叫 XMLHttpRequest；而在 Internet Explorer 中，它是 Microsoft XMLHttpRequest 或 Msxml2.XMLHTTP 的 ActiveX 组件（不过在 IE 7 中已更名为 XMLHttpRequest）。

从上面的叙述可见：Ajax 应用程序必须是由客户端和服务端一同合作的应用程序，JavaScript 是撰写 Ajax 程序的客户端语言，XML 则是请求或回应时建议使用的交换信息的格式。

4.4.2 Ajax 基础

1. XMLHttpRequest 对象

在 JavaScript 中，XMLHttpRequest 对象提供客户端与 HTTP 服务器异步通信的协议。通过该协议，Ajax 可以使页面像桌面应用程序一样，只同服务器进行数据层面的交换，而不用每次都刷新界面，也不用每次将数据处理工作都提交给服务器去做，既减轻了服务器的负担，又加快了响应速度。

在 Ajax 应用程序中，如果使用的是 Mozilla、Firefox 或 Safari，可通过 XMLHttpRequest 对象发送非同步请求；如果使用的是 IE 6 及之前的版本，则使用 ActiveX 对象来发送请求。

为了满足各种不同浏览器的兼容性，必须先进行测试，取得 XMLHttpRequest 或 ActiveX 对象，形如下面的代码：

```
var xmlhttp;  
function createXMLHttpRequest(){  
    if(window.XMLHttpRequest){  
        xmlhttp=new XMLHttpRequest();  
    }  
    else if(window.ActiveXObject){  
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
    }  
}
```

创建了 XMLHttpRequest 对象后，通过在 JavaScript 脚本中调用它的方法（见表 4.10）和属性（见表 4.11），实现 Ajax 的功能。

表 4.10 XMLHttpRequest 对象的方法

方 法 名	描 述
abort()	停止当前请求
getAllResponseHeaders()	将 HTTP 请求的所有响应首部作为键/值对返回
getResponseHeader("header")	返回指定首部的字符串值
open("method","url",[asyncFlag[, "userName" [, "password"]]])	建立对服务器的调用，method 参数可以是 GET、POST 或 PUT，url 参数可以是相对或绝对 URL。该方法还有 3 个可选参数： asyncFlag = 是否非同步标记 username = 用户名 password = 密码
send(content)	向服务器发送请求
setRequestHeader("header","value")	把指定首部设置为所提供的值，在调用该方法之前必须先调用 open 方法

表 4.11 XMLHttpRequest 对象的属性

属 性 名	描 述
onreadystatechange	状态改变事件触发器，每个状态改变都会触发这个事件触发器
readyState	对象状态： 0 = 未初始化 1 = 正在加载 2 = 已加载 3 = 交互中 4 = 完成
responseText	服务器的响应，字符串
responseXML	服务器的响应，XML。该对象可以解析为一个 DOM 对象
status	服务器返回的 HTTP 状态码
statusText	HTTP 状态码的相应文本

2. Ajax 原理

Ajax 利用浏览器与服务器之间的一个通道来“暗中”完成数据提交或请求。具体方法是：页面的脚本程序通过浏览器提供的控件完成数据的提交和请求，并将返回的数据由 JavaScript 处理后展现到页面上。整个过程由浏览器、JavaScript 和 JSP 共同完成，不同的浏览器对 Ajax 有不同的支持方法，但对于 Web 服务器来说没有任何变化，因为浏览器和服务器之间的这个通道依然是基于 HTTP 请求和响应的，浏览器正常的请求和 Ajax 请求对于 Web 服务器来说并没有任何区别。图 4.16 说明了 Ajax 的请求和响应过程。

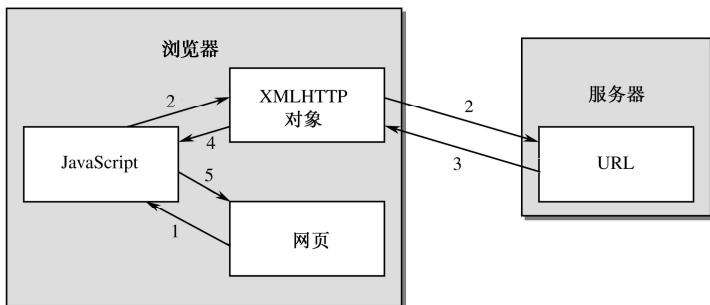


图 4.16 Ajax 的请求和响应过程

Ajax 的请求和响应过程如下。

- (1) 网页调用 JavaScript 程序。
- (2) JavaScript 利用浏览器提供的 XMLHttpRequest 对象向 Web 服务器发送请求。
- (3) 请求的 URL 资源处理后将返回结果给浏览器的 XMLHttpRequest 对象。
- (4) XMLHttpRequest 对象调用事先设置的处理方法。
- (5) JavaScript 方法解析返回的数据，用返回的数据更新页面。

3. Ajax 适用场合

Ajax 虽然是一个好的技术，但它也不是万能的。在适宜的场合使用 Ajax，才能充分发挥其长处，改善系统性能和用户体验，绝不能为了技术而滥用。Ajax 的特点在于异步交互、动态更新 Web 页，因此它适于交互较多、频繁读取数据的 Web 应用。

下面列举几个 Ajax 常用的场合。

(1) 数据验证

在填写表单内容时，有时需要保证数据的唯一性（如新用户注册时填写的用户名），因此必须对用户输入的内容进行验证。使用 Ajax 技术，可以由 XMLHttpRequest 对象发出验证请求，根据返回的 HTTP 响应判断验证是否成功，整个过程不需要弹出新窗口，也不需要将整个页面提交到服务器端，快速而又不加重服务器负担。在这种情况下，Ajax 技术是很好的选择。

(2) 按需取数据

分类树或树形结构在 Web 应用系统中的使用非常普遍，一次性将分类结构中的数据全部读取出来并写入数组，然后根据用户的操作需求，用 JavaScript 来控制节点的呈现。但是如果用户不对分类树进行操作，或者只对分类树中的一部分数据进行操作，那么读取的数据就会成为垃圾资源。Ajax 技术改进了分类树的实现机制：在初始化页面时，只获取根部分类数据并显示它们，当用户单击根部分类的某个子节点时，页面通过 Ajax 向服务器请求当前分类所属的子分类的数据……以此类推，页面会根据用户的操作，仅仅向服务器请求它所需要的数据，减少了数据加载量，实现页面的局部刷新。

(3) 自动更新页面

在 Web 页面上有很多数据变化十分迅速,如股市、天气预报等,而 Ajax 解决了页面自动更新的问题。页面加载后,通过 Ajax 引擎在后台定时向服务器发送请求,查看是否有最新的消息。如果有,则加载新的数据,并且在页面上动态更新,然后通过一定的方式通知用户。这样既避免了用户不断手动刷新页面的不便,也不会因页面定时重复刷新而造成资源浪费。

4.4.3 Ajax 应用实例

在前面的【例 4.2】中,需等到提交后才能验证表单的有效性,然后提醒用户完善输入内容。这样用户就需要等待一个页面刷新的阶段,显然不能令人满意。Ajax 的出现正好解决了这个问题,其无刷新机制使得用户在注册时能对所填写内容的有效性做出即时的判断,给用户以全新的体验。

【例 4.6】利用 Ajax 技术重新实现之前 4.1.6 节的【例 4.2】校验用户注册页面中的表单内容是否为空的功能。

(1) 方案选择和准备

针对 Ajax 技术,在 Java EE 领域有不少现成的解决方案,如 DWR、AjaxAnywhere、JSON-RPC-Java 等。DWR 是开源框架,借助它,开发人员甚至无须具备专业的 JavaScript 知识就能轻松实现 Ajax,使 Ajax 应用更加“平民化”。

从 DWR 官方网站 <http://directwebremoting.org/dwr/> 下载 DWR 开发包 dwr.jar (Version 2.0.9)。

登录 <http://struts.apache.org/> 下载 Struts 2 完整版(本书使用最新发布的 Struts 2.3.1.1),将下载的 Zip 文件解压缩,取 lib 目录下的 commons-logging-1.1.1.jar 包。

(2) 建立项目

打开 MyEclipse,建立 Web 项目,命名为“4_6usrchkbyAjaxDwr”。将第(1)步准备好的 DWR 开发包 dwr.jar (Version 2.0.9) 和 commons-logging-1.1.1.jar 复制到项目的 WEB-INF\lib 文件夹下。

有关 Web 项目的创建及其结构,经过第 2 章的学习,相信大家一定很清楚了,这里仅列出本项目的目录结构,如图 4.17 所示,以供读者参考。

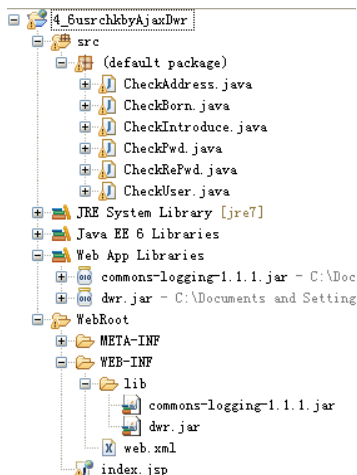


图 4.17 项目生成的目录

(3) 编写客户端程序

打开图 4.17 目录树中的文件 index.jsp，将其原来的内容全部删除，自己编写代码如下：

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head>
    <title>Ajax 应用</title>
</head>
    <script type="text/javascript">
        var xmlhttp;
        //创建 XMLHttpRequest 对象
        function createHttpRequest(){
            if(window.ActiveXObject){
                xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            else if(window.XMLHttpRequest){
                xmlhttp = new XMLHttpRequest();
            }
        }
        /*验证用户名*/
        function usernameCheck(){
            ...
            xmlhttp.onreadystatechange = processor_usrchk;
            ...
        }
        function processor_usrchk(){
            ...
        }
        /*验证密码*/
        function passwordCheck(){
            ...
            xmlhttp.onreadystatechange = processor_pwdchk;
            ...
        }
        function processor_pwdchk(){
            ...
        }
        /*验证确认密码*/
        function repasswordCheck(){
            ...
            xmlhttp.onreadystatechange = processor_repwdchk;
            ...
        }
        function processor_repwdchk(){
            ...
        }
        /*验证出生日期*/
```

```

function bornCheck(){
    ...
    xmlhttp.onreadystatechange = processor_bornchk;
    ...
}
function processor_bornchk(){
    ...
}
/*验证地址*/
function addressCheck(){
    ...
    xmlhttp.onreadystatechange = processor_addresschk;
    ...
}
function processor_addresschk(){
    ...
}
/*验证自我介绍*/
function introduceCheck(){
    ...
    xmlhttp.onreadystatechange = processor_introducechk;
    ...
}
function processor_introducechk(){
    ...
}
</script>
<body>
<meta http-equiv="content-type" content="text/html; charset=gb2312">
<form action="" method="post" name="form1" onsubmit="">
<table width="409" border="1">
<tr>
<td>用户名: </td>      <!--设置用户名的表单-->
<!-- 当输入框改变时执行 usernameCheck()函数 -->
<td><input type="text" name="username" onchange="usernameCheck()"></td>
</tr>
<tr>
<td>密码: </td>      <!--设置用户登录密码的表单-->
<!-- 当输入框改变时执行 passwordCheck()函数 -->
<td><input type="password" name="password" onchange="passwordCheck()"></td>
</tr>
<tr>
<td>确认: </td>      <!--设置确认密码的表单-->
<!-- 当输入框改变时执行 repasswordCheck()函数 -->
<td><input type="password" name="repassword"
onchange="repasswordCheck()"></td>

```

```

        </tr>
        <tr>
            <td>出生: </td>          <!--设置出生日期的表单-->
            <!-- 当输入框改变时执行 bornCheck()函数 -->
            <td><input type="text" name="born" onchange="bornCheck()"></td>
        </tr>
        <tr>
            <td>地址: </td>          <!--设置地址的表单-->
            <!-- 当输入框改变时执行 addressCheck()函数 -->
            <td><input type="text" name="address" onchange="addressCheck()"></td>
        </tr>
        <tr>
            <td>介绍: </td>          <!--设置介绍的表单-->
            <!-- 当输入框改变时执行 introduceCheck()函数 -->
            <td><textarea name="introduce" rows="5" id="introduce"
                                onchange="introduceCheck()"></textarea></td>
        </tr>
    </table>
    <input type="submit" name="Submit1" value="注册">
    <input type="reset" name="Submit2" value="重置">
</form>
</body>
</html>

```

在上面的代码中，对应页面表单的每个输入框，都由 `onchange` 事件触发一个 JavaScript 函数去执行相应字段的验证过程，比如验证用户名的函数 `usernameCheck()` 的代码如下：

```

/*验证用户名*/
function usernameCheck(){
    //得到用户填写的用户名
    var username=document.all.username.value;
    createHttpRequest();
    //将状态触发器绑定到一个函数
    xmlHttp.onreadystatechange = processor_usrchk;
    //通过 get 方法向指定的 URL 即 Servlet 对应 URL 建立服务器的调用
    xmlHttp.open("get","CheckUser?username="+username);    //由一个 HttpServlet 实现验证
    //发送请求
    xmlHttp.send(null);
}
function processor_usrchk(){
    var responseContext;
    //如果响应完成
    if(xmlHttp.readyState == 4){
        //如果返回成功
        if(xmlHttp.status == 200){
            //取得响应内容
            responseContext = xmlHttp.responseText;
            //如果用户名检查为空

```

```
        if(responseContext.indexOf("true")==-1){
            alert("请输入用户名");
        }
    }
}
```

表单中其他输入框的验证函数与此基本相同，就不一一列出了。可以看到，用户名的验证由一个名为“CheckUser”（加黑处）的 `HttpServlet` 来实现。

（4）编写后台用于验证的 Servlet 类

还是以用户名的验证为例，在项目 `src` 下创建类文件 `CheckUser.java`，代码如下：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CheckUser extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        //取得用户填写的用户名
        String username=request.getParameter("username");
        //设置响应内容
        String responseContext="true";
        if(username=="")
        {
            responseContext="false";
        }
        //将处理结果返回给客户端
        out.println(responseContext);
        out.flush();
        out.close();
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request,response);
    }
}
```

在介绍 Servlet 的时候说过，有 Servlet 文件存在就要进行相应的配置。打开项目 `WebRoot\WEB-INF` 下的 `web.xml` 文件，在其中添加如下配置：

```
<servlet>
    <servlet-name>CheckUser</servlet-name>
    <servlet-class>CheckUser</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>CheckUser</servlet-name>
    <url-pattern>/CheckUser</url-pattern>
</servlet-mapping>
```

本例其余 Servlet 的编码及配置与此类同，请读者参照着自己完成。

(5) 运行程序

部署运行，可以验证实际效果。先填写表单的各项内容，然后故意将用户名删去，单击页面的任意处，都会提示“请输入用户名”，如图 4.18 所示。

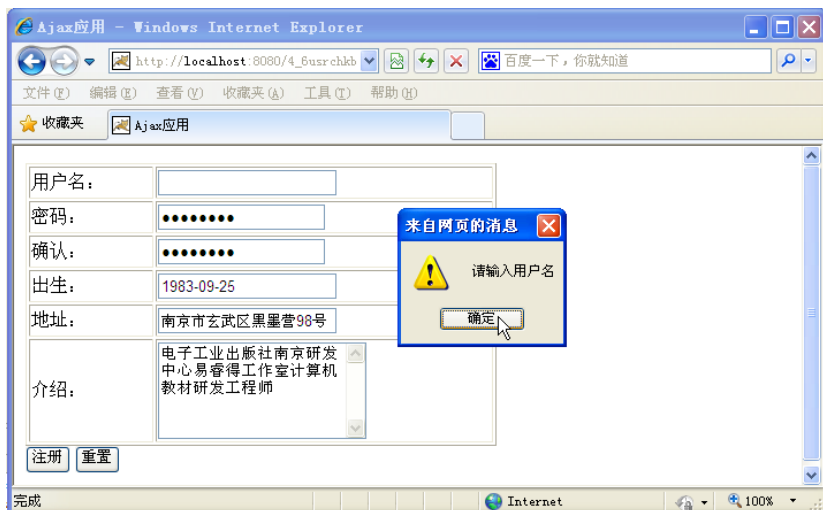


图 4.18 用 Ajax 验证表单内容是否为空

对比之前【例 4.2】的图 4.7，两者运行效果一样，但本例应用了 Ajax，优点在于：无须单击“注册”按钮就可即时验证页面上的输入内容，提高了响应速度，改善了用户体验。

4.5 上机练习

1. 参照【例 4.3】的例子制作隐式菜单，如图 4.19 所示。

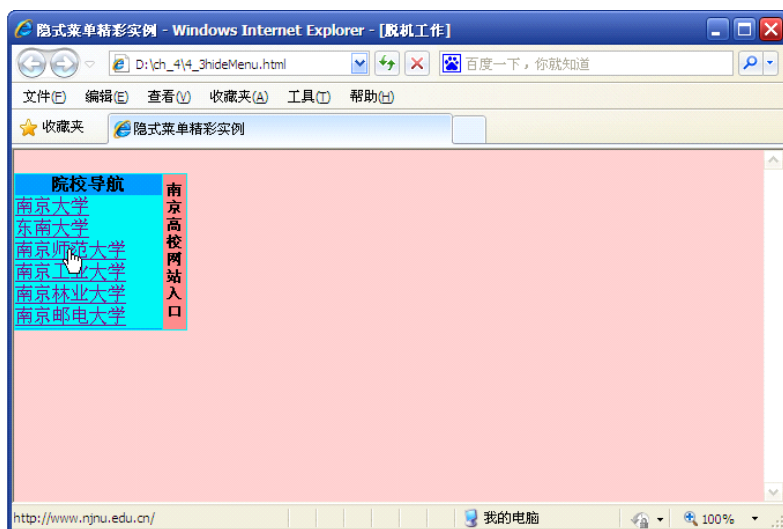


图 4.19 显示子菜单、打开链接页

- (1) 在菜单中加入自己的学校链接, 运行程序, 单击链接, 测试其可访问性。
 - (2) 试在一级子菜单下增加二级子菜单, 链接指向校内各院系首页。
2. 设计网页, 文件名为 4_4yogdna.html, 页面显示如图 4.20 所示 (与图 4.11 相同)。

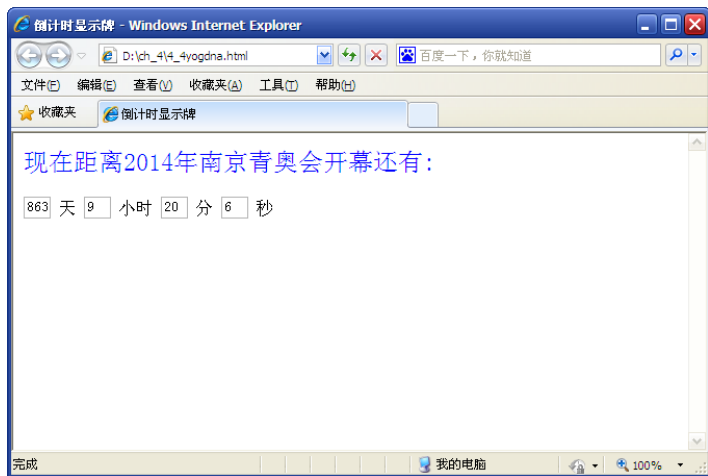


图 4.20 青奥会倒计时牌的运行效果

2014 年第二届夏季青年奥林匹克运动会将于 2014 年 8 月 16 日至 28 日在中国南京举办。请读者试修改上述代码, 计算 2014 年 8 月 16 日是星期几, 青奥会闭幕日 (28 日) 又是星期几。

3. 仿照【例 4.5】制作图像自由运动效果, 有兴趣的读者可将热带观赏鱼照片换成其他动物或卡通、动漫造型的图片, 尝试不一样的效果。

4. 分别用【例 4.2】和【例 4.6】的两种不同方法, 实现校验用户注册页的表单内容是否为空的功能。运行程序, 通过对比体会 Ajax 技术的优点。

JSP 服务器对象应用

JSP 服务器对象是指在 JSP 中**内置的**、无须定义就可以在网页中直接使用的对象。这些对象对页面起简化作用，因为它们不需要由 JSP 程序员进行实例化，而是由容器实现和管理，并且在所有的 JSP 页面中都能够使用。

5.1 内置对象及其作用

5.1.1 JSP 内置对象

JSP 内置对象有 Request、Response、Session、Application、Page、PageContext、Out、Config 和 Exception **九个对象**，它们在 JSP 页面初始化时生成，可以灵活利用这些对象制作多种 JSP 应用程序。

在上述九个对象中，**Request、Response 和 Session** 这三个对象体现了服务器与客户端进行交互通信的控制，是最基本、最重要的对象。除了以上三个基本对象，比较重要且常用的还有 Application 和 Out 两个对象。

1. Request 对象

Request 对象包含了来自客户端的请求信息，如请求的来源、标头、Cookie 及与请求相关的参数值等。请求经 Servlet 容器处理后，由 Request 对象进行封装，然后作为_jspService()方法的一个参数由容器传递给 JSP 页面。客户端可以通过 HTML 表单或者在网页地址后带参数的方法提交数据，再用 Request 对象的相关方法来获取提交的各种数据。

2. Response 对象

Response 对象和 Request 对象的性质相反，它所包含的是服务器向客户端做出的应答信息。JSP 引擎会根据客户端的请求信息建立一个预设的 Response 回应对象，然后传入_jspService()方法中，提供给客户端浏览器某些参考信息。

3. Session 对象

Session 对象用于表示当前的某个用户会话，Session 中保存的对象在当前用户链接的所有页面中都可以被访问到。一般使用 Session 对象存储用户登录网站的信息，当用户在页面之间跳转时，存储在 Session 对象中的变量不会被清除。

4. Application 对象

不同用户的 Session 对象互不相同，但有时候用户之间可能需要共享一个对象。Web 服务器启动后，就产生了一个共享对象——Application 对象。任何客户端在访问服务目录的各个页面时，Application 对象都是同一个，直到关闭服务器后，Application 对象才会消失。

5. Out 对象

Out 对象向 Web 浏览器输出各种数据类型的内容，并且管理应用服务器上的输出缓冲区，它是 JSP 编程过程中经常用到的一个对象，在前几章的编程实践中，大家已经接触到很多次了。

5.1.2 客户端/服务器交互

Request 对象和 Response 对象的结合，可以使 JSP 更好地实现客户端与服务器的信息交互。

用户在客户端浏览器中发出的请求信息被保存在 Request 对象中并发送给 Web 服务器，JSP 引擎根据 JSP 文件的指示处理 Request 对象，或者根据实际需要将 Request 对象转发给由 JSP 文件所指定的其他服务器端组件，如 Servlet 组件、JavaBean 组件或 EJB 组件等。处理结果则以 Response 对象的方式返回给 JSP 引擎，JSP 引擎和 Web 服务器根据 Response 对象最终生成 JSP 页面，返回给客户端浏览器，这也是用户最终看到的内容。

客户端与服务器信息交互的流程如图 5.1 所示。

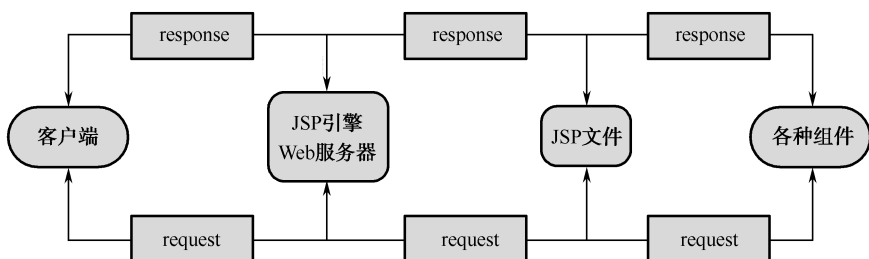


图 5.1 客户端与服务器信息的交互流程

由于客户端和服务器之间最常用的通信协议是 HTTP（也可以使用特定的私有协议），因此，Response 对象在 JSP 响应客户请求时的作用是很大的。

5.1.3 对通信的控制

Request、Response 和 Session 三者是 JSP 内置对象中的重要对象，它们直接主导着客户端/服务器双方整个交互通信的控制。

当客户端用户打开浏览器，在地址栏中输入服务器 Web 页面的地址后，就会显示服务器上的网页。浏览器从 Web 服务器上获得网页的过程实际上就是使用 HTTP 向服务器发送一个请求，服务器则接收这个来自客户端的请求。这中间 Request、Response 和 Session 三个对象的通信过程如图 5.2 所示。

如图 5.2 所示，JSP 通过 Request 对象控制客户端浏览器的请求；通过 Response 对客户端浏览器进行响应；而 Session 则维持这个反反复复的会话过程中需要传递的数据信息。

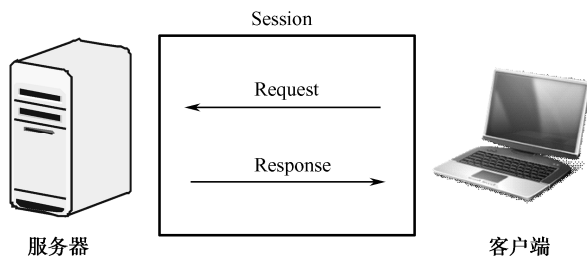


图 5.2 三个内置对象的通信过程

5.2 请求对象：Request

Request 对象可以对在客户请求中给出的信息进行访问，它是 `HttpServletRequest` 的一个子类，其作用域就是一次 Request 请求。

5.2.1 获取请求参数

通常情况下，Web 应用程序需要用户与网站交互。用户填写表单后，要把数据提交给服务器处理。Request 对象的 `getParameter()` 方法，可以用来获取用户提交的数据。

获取请求参数的使用格式如下：

```
String name=request.getParameter("name");
```

代码说明：

参数 `name` 与 form 表单中的 `name` 属性对应，或者与提交链接的参数名对应，如果参数值不存在，则返回 `null` 值，该方法的返回值类型是 `String`。

【例 5.1】 `getParameter()` 方法的应用。

输入以下内容，以 `5_1reqGetPar.html` 作为文件名保存：

```
<%@page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title>request 对象使用实例 1 </title>
</head>
<body bgcolor="#bbbfef">
    <form method="post" action="/5_1reqGetPar.jsp">
        <table align="center" bgcolor="silver" width="300">
            <tr bgcolor="#0038ff">
                <td colspan="2" align="center">
                    <font color="white">登录表单</font>
                </td>
            </tr>
            <tr>
                <td width="30%" height="30" align="right">登录名: </td>
                <td width="70%" align="left">
                    <input type="text" name="uname" size="20">
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

```

        </td>
    </tr>
    <tr>
        <td width="30%" height="30" align="right">密码: </td>
        <td width="70%" align="left">
            <input type="password" name="password" size="20">
        </td>
    </tr>
    <tr>
        <td colspan="2" align="center">
            <input type="submit" value="提交">
            <input type="reset" value="重填">
        </td>
    </tr>
</table>
</form>
</body>
</html>

```

在 5_1reqGetPar.html 文件中设定了页面的结构, 其中包括两个文本框和两个按钮。下面是 5_1reqGetPar.jsp (指向页面) 文件的代码:

```

<%@ page language="java" contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.*" %>
<html>
<head>
    <title> request 对象使用实例 1 </title>
</head>
<body bgcolor="#bbbf"ff">
    <form>
        你刚才输入的内容是: <br>
        <%
            request.setCharacterEncoding("gb2312");
            String userName = request.getParameter("uname");
            String passWord = request.getParameter("password");
            out.print("登录名: " + userName + "<br>");
            out.print("密码: " + passWord + "<br>");
        %>
    </form>
</body>
</html>

```

运行程序, 如图 5.3 所示。

输入登录名和密码, 单击“提交”按钮后, 运行结果如图 5.4 所示。



图 5.3 登录表单

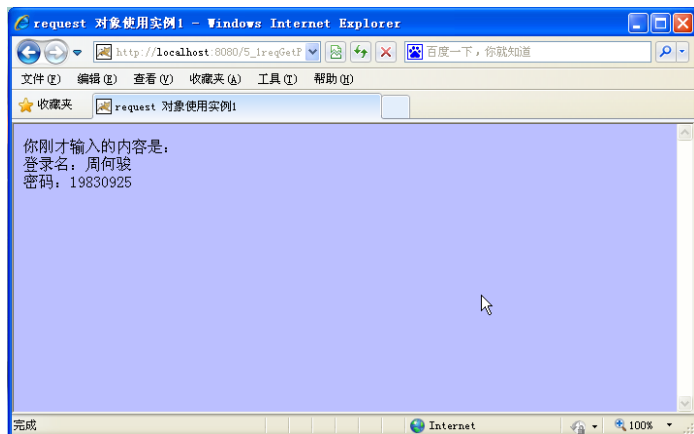


图 5.4 获取登录名、密码两个参数

5.2.2 设置/获取属性

通常情况下,在进行请求转发操作时,会把一些属性带到转发后的页面处理。这时,就可以使用 Request 对象的 `setAttribute()` 方法将属性设置在 Request 范围内存取。

在 Request 作用域中,设置转发属性的方法的格式如下:

```
request.setAttribute("key", value);
```

代码说明:

参数 `key` 为 String 类型的属性名。在转发后的页面获取数据时,通过这个属性名来获取;参数 `value` 为 Object 类型的属性值,需要保存在 Request 范围内的数据。

在 Request 作用域中,获取转发属性的方法的格式如下:

```
Object object=request.getAttribute("name");
```

代码说明:

在页面使用 Request 对象的 `request.setAttribute("key",value)` 方法,可以把属性 `key` 设置在 Request 范围内。请求转发后的页面使用 `request.getAttribute("name")` 方法就可以获取属性值 `value`,该方法的返回值是 Object。

在 Request 作用域中，获取所有属性的名称集的方法的格式如下：

```
request.getAttributeNames();
```

代码说明：

该方法的返回值是枚举类型（Enumeration）数据。

【例 5.2】setAttribute()/getAttribute()方法的应用。

输入以下内容，以 5_2reqAttr.jsp 作为文件名保存：

```
<%@ page language="java" contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.*" %>
<html>
<head>
    <title> request 对象使用实例 2</title>
</head>
<body>
    <font size="6" color="blue">
        <%
            request.setAttribute("TestString","欢迎访问南京师范大学");
        %>
        <%=request.getAttribute("TestString")%>
    </font>
</body>
</html>
```

运行结果如图 5.5 所示。

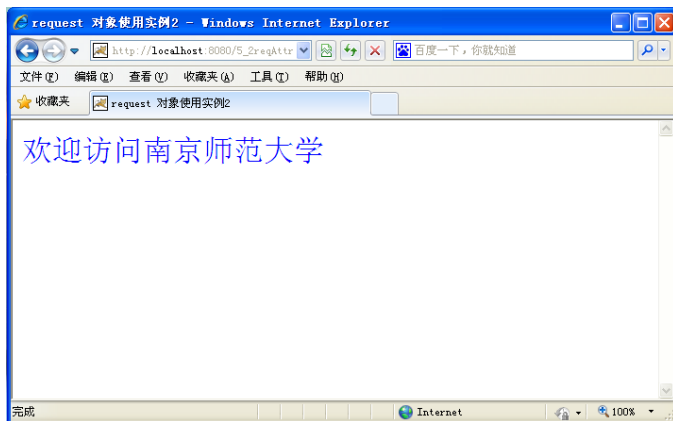


图 5.5 获取属性

5.2.3 获取其他信息

Request 对象还有一些其他的方法，可以用于获取组成 JSP 页面的客户端、客户端请求等的更多信息。

【例 5.3】应用 Request 的方法输出更多信息。

输入以下内容，以 5_3reqMoreInfo.jsp 作为文件名保存：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
```

```

<head>
    <title> request 对象使用实例 3 </title>
</head>
<body bgcolor="#bbbfff">
    <font size="3">
        <h2>request 对象的应用 3</h2>
        <hr>
        <b>获得 http 头文件中 host 的值:</b><br>
            <%= request.getHeader("host") %><br>
        <b>获得服务器的名称:</b><br>
            <%= request.getServerName() %><br>
        <b>获得服务器的端口号:</b><br>
            <%= request.getServerPort() %><br>
        <b>获得客户端的 ip 地址 :</b><br>
            <%= request.getRemoteAddr() %><br>
        <b>获得客户端的主机名:</b><br>
            <%= request.getRemoteHost() %><br>
        <b> 获得客户使用的协议名称:</b><br>
            <%= request.getProtocol() %><br>
        <b>获得客户提交信息的方式:</b><br>
            <%= request.getMethod() %><br>
        <b>获得接收客户提交信息的页面:</b><br>
            <%= request.getServletPath()%><br>
        <b>获得请求中的字符编码方式:</b><br>
            <%= request.getCharacterEncoding()%><br>
        <b>获得发出请求字符串的客户端地址:</b><br>
            <%= request.getRequestURI()%><br>
    </font>
</body>
</html>

```

运行结果如图 5.6 所示。

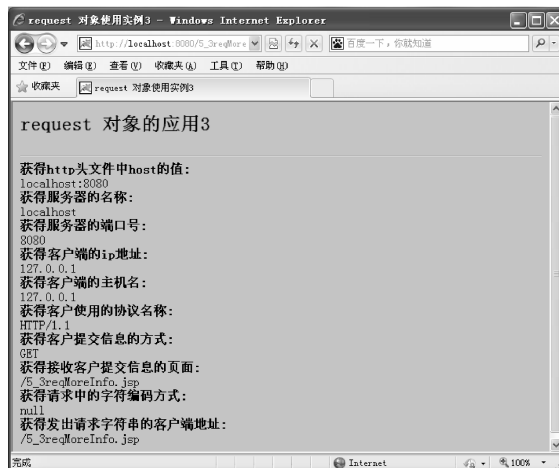


图 5.6 获取更多信息

5.2.4 Request 方法一览

以上介绍了 Request 对象最常用的几个方法，还有很多其他的方法，现将 Request 的主要方法整理并列于表 5.1 中，供读者编程时参考。

表 5.1 Request 对象的主要方法

方法名称	含义
getParameter(String name)	以字符串的形式返回客户端传来的某个请求参数的值，该参数由 name 指定。当传递此方法的参数名没有实际参数与之对应时，返回 null。另外，当一个参数含有多个值时最好不要使用这个方法
getParameterValue(String name)	以字符串数组的形式返回指定参数的所有值
getParameterNames()	返回客户端传送给服务器端的所有参数名，结果集是一个 Enumeration（枚举）类的实例。当没有任何参数时，返回 null
getAttribute(String name)	返回 name 指定的属性值，若不存在指定的属性，则返回 null
setAttribute(String name, Object obj)	设置名字为 name 的 Request 参数的值为 obj
getCookies()	返回客户端的 Cookies 对象，结果是一个 Cookie 数组
getHeader(String name)	获得 HTTP 协议定义的传送文件头信息，例如 request.getHeader("User-Agent")，其含义为：返回客户端浏览器的版本号、类型
getDateHeader()	返回一个 Long 类型的数据，表示客户端发送到服务器的头信息中的时间信息
getHeaderName()	返回所有 request header 的名字，结果集是一个 Enumeration（枚举）类的实例。得到名称后就可以使用 getHeader、getDateHeader 等得到具体的头信息
getServerPort()	获得服务器的端口号
getServerName()	获得服务器的名称
getRemoteAddr()	获得客户端的 IP 地址
getRemoteHost()	获得客户端的主机名，如果该方法失败，则返回客户端的 IP 地址
getProtocol()	获得客户端向服务器端传送数据所依据的协议名称
getMethod()	获得客户端向服务器端传送数据的方法
getServletPath()	获得客户端所请求的脚本文件的文件路径
getCharacterEncoding()	获得请求中的字符编码方式
getSession(Boolean create)	返回和当前客户端请求相关联的 HttpSession 对象。如果当前客户端请求没有和任何 HttpSession 对象关联，那么如果 create 变量为 true，则创建一个 HttpSession 对象并返回，反之返回 null
getQueryString()	返回查询字符串，该字符串由客户端以 GET 方式向服务器端传送。查询字符串出现在页面请求“?”的后面，如 http://www.njnu.edu.cn/hello.jsp?name=Jack
getContextPath()	返回请求 URI 部分，表示请求的应用程序环境
getRequestURI()	获得发出请求字符串的客户端地址
getPathInfo()	返回任何额外的路径信息，这些信息与服务器小程序路径、查询字符串之间的 URL 相关联
getContentType()	获取客户端请求的 MIME 类型。如果无法得到该请求的 MIME 类型，则返回-1

5.3 响应对象：Response

Response 对象实现了 HttpServletResponse 接口，为 HttpServletResponse 类或其子类的一个对象，它可以对客户的请求做出动态的响应。Response 对象用于将服务器端数据发送到客户端，数据的发送可通过 HTTP 文件头信息、页面重定向及缓冲区等方式进行。

5.3.1 发送 HTTP 文件头

响应客户端最简便的方式是使用 `Response` 对象的 `setHeader()` 方法，该方法设定一个指定名字的 HTTP 文件头，通过这个文件头向客户端传达响应信息。

`setHeader()` 方法的格式如下：

```
response.setHeader("name",value);
```

下面看两个典型例子。

【例 5.4】 利用 `setHeader()` 方法实现网页自动更新。

输入以下内容，以 `5_4rspHeader.jsp` 作为文件名保存：

```
<%@page contentType="text/html;charset=gb2312"%>
<%@page import="java.util.*"%>
<html>
<head>
    <title> setheader 方法的应用</title>
</head>
<body bgcolor="#fffccf">
    <font size="3">
        <p><b>本例用来说明用 response 对象的 setheader 方法实现网页自动更新</b></p>
        <br>
        <b>现在时间为： </b>
        <%
            response.setHeader("refresh","5");           //页面每隔 5 秒刷新一次
            out.println(new Date());
        %>
    </font>
</body>
</html>
```

运行结果如图 5.7 所示。

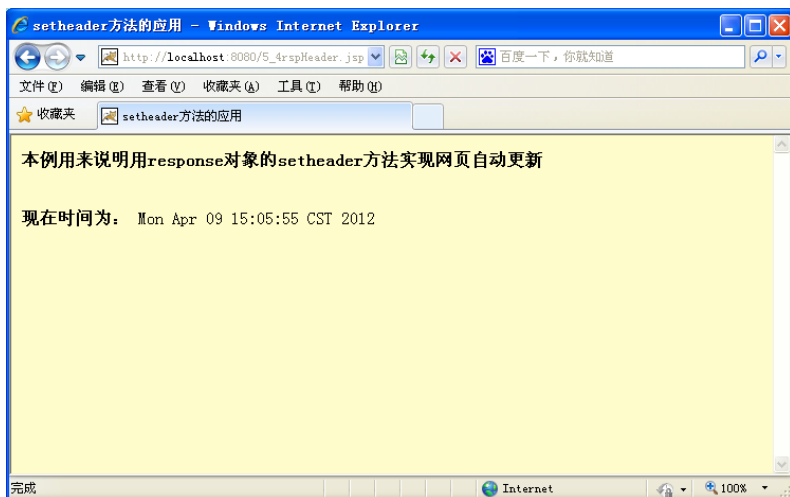


图 5.7 利用 HTTP 文件头自动更新网页

【例 5.5】 利用 `setHeader()` 方法将当前页面保存为 Word 文档。

输入以下内容，以 `5_5rspWord.jsp` 作为文件名保存：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
    <title>response 对象的应用</title>
</head>
<body>
    <h2>
    <p>要将当前的 jsp 页面保存为 word 文档，</p>
    <p>请点击此按钮。</p>
    <form action="" method="get" name="fl">
        <input type="submit" value=" Save As Word " name="sub1">
    </form>
<%
    String str=request.getParameter("sub1");
    if(str==null)
    {
        str="";
    }
    if(str.equals(" Save As Word "))
    {
        response.setContentType("application/msword;charset=gb2312");
        response.setHeader("Content-Disposition","attachment;filename=5_5rspWord.doc");
    }
%>
</body>
</html>
```

说明：

使用 `getParameter()` 方法取得按钮的值，若是“Save As Word”，则说明用户已经按下了按钮。设置 `Response` 对象的 `setContentType` 属性为“`application/msword;charset=gb2312`”，用来改变 `ContentType` 的属性值，并且设置了其 `setHeader` 的属性，使下载的 word 文档的默认文件名为“`5_5rspWord.doc`”。这时，JSP 页会动态地改变 `ContentType` 值，浏览器就会提示用户用 Word 格式来保存当前页面。

运行程序，浏览器显示页面如图 5.8 所示。

单击“Save As Word”按钮后，会出现 Windows 的“文件下载”对话框，如图 5.9 所示。

若单击“打开”按钮，则直接打开该 Word 文档；若单击“保存”按钮，在指定保存路径后，就可以看到该路径下有文件“`5_5rspWord.doc`”出现。打开文件“`5_5rspWord.doc`”，显示 Word 文档如图 5.10 所示，与图 5.8 浏览器显示页的内容完全一样。

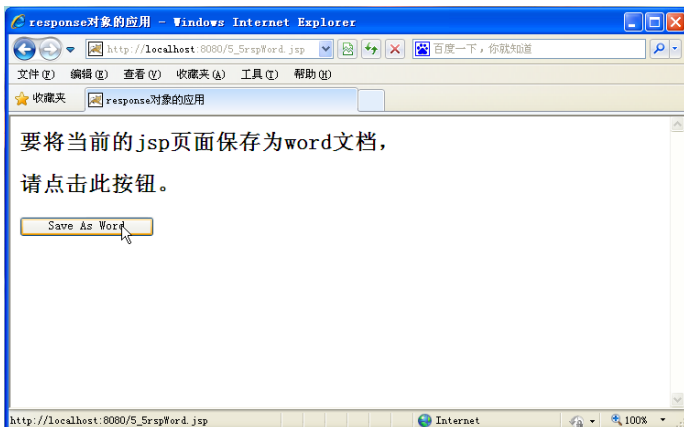


图 5.8 浏览器显示的页面

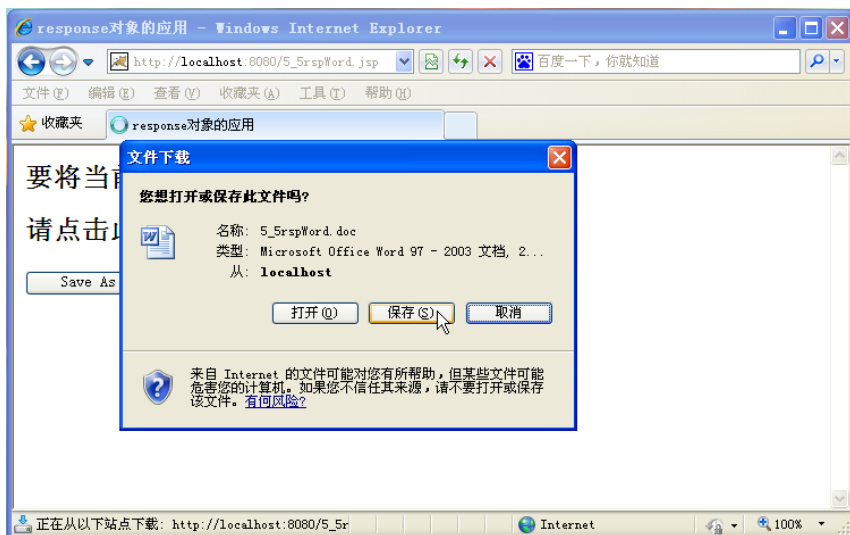


图 5.9 弹出“文件下载”对话框

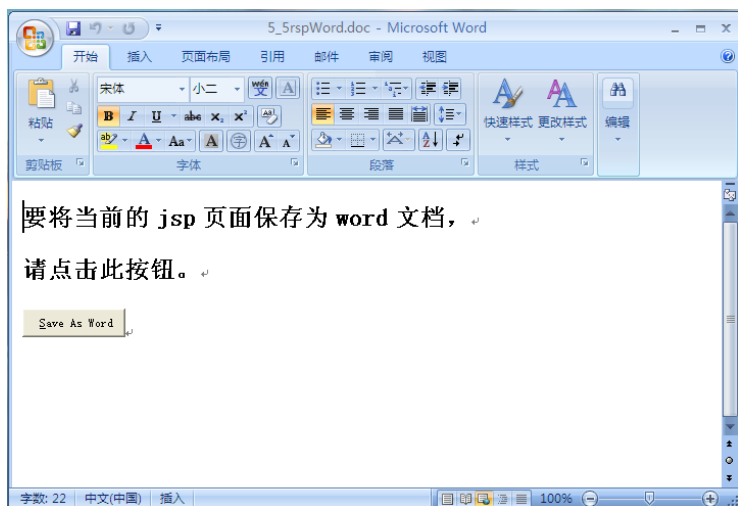


图 5.10 Word 文档的内容

5.3.2 页面重定向

JSP 页面可以使用 `Response` 对象中的 `sendRedirect()` 方法将客户请求重定向到另一个不同的页面资源。例如，客户端重定向到 `easybooks.jsp` 页面的代码如下：

```
response.sendRedirect("easybooks.jsp");
```

JSP 页面还可以使用 `Response` 对象中的 `sendError()` 方法指明一个错误状态，该方法接收一个错误及一条可选的错误信息。该信息将在内容主体上返回给客户。例如，将客户端请求重定向到一个在内容主体上包含了出错信息页面的代码如下：

```
response.sendError(500, "请求页面存在错误");
```

上述两个方法都会中止当前的请求和响应。如果 HTTP 响应已经提交给客户端，则不会调用这些方法。

`Response` 对象中用于重定向的方法如下。

- `sendError(int number)` 方法：使用指定的状态码向客户发送错误响应。
- `sendError(int *number, String msg)` 方法：使用指定的状态码和描述性消息向客户发送错误响应。
- `sendRedirect(String location)` 方法：指定的重定向位置 URL 并向客户发送重定向响应，可以使用相对 URL。

5.3.3 缓冲区输出

缓冲可以有效地在服务器与客户端之间传输内容。`HttpServletResponse` 对象为支持 `jspWriter` 对象而启用了缓冲区配置。`Response` 对象中的 `getBufferSize()` 方法的返回值用于获取 JSP 页面的当前缓冲区容量；`Response` 对象中的 `setBufferSize()` 方法允许 JSP 页面为响应的主体设置一个首选的输出缓冲区容量。容器使用的实际缓冲区容量至少要等于输出缓冲区的容量。如果要设置缓冲区容量，则必须在向响应中写入内容之前设置，否则 JSP 容器将产生异常。

`Response` 对象用于响应缓冲的方法如下。

- `flushBuffer()` throws `IOException`：强制把缓冲区中的内容发送给客户端。
- `getBufferSize()`：返回响应所使用的实际缓冲区大小，如果没使用缓冲区，则该方法返回 0。
- `setBufferSize(int size)`：为响应的主体设置首选的缓冲区大小。
- `boolean isCommitted()`：表示响应是否已经提交，提交的响应已经写入状态码。
- `reset()`：清除缓冲区存在的任何数据，同时清除状态码。

【例 5.6】Response 缓冲区的使用。

要求：在页面中首先通过 `Response` 对象调用 `getBufferSize()` 方法，显示当前页面缓冲区的大小，之后调用 `Response` 对象中的 `flushBuffer()` 方法，强制把缓冲区中的内容发送给客户端，最后再显示当前缓冲区的大小。

输入以下内容，以 `5_6rspBuf.jsp` 作为文件名保存：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
```

```

<title> response 对象使用缓冲区 </title>
</head>
<body bgcolor="#bbbf"ff">
  <table border="1">
    <tr>
      <td height="30" colspan="2" align="center">缓冲区设置之前</td>
    </tr>
    <tr>
      <td width="131" height="30">当前缓冲区大小: </td>
      <td width="223"><%=response.getBufferSize()%></td>
    </tr>
    <tr>
      <td height="30">输出的内容是否提交: </td>
      <td><%=response.isCommitted()%></td>
    </tr>
    <tr>
      <td height="30" colspan="2" align="center">缓冲区设置之后
        <%=response.flushBuffer();%></td>
    </tr>
    <tr>
      <td height="30">输出的内容是否提交: </td>
      <td><%=response.isCommitted()%></td>
    </tr>
    <tr>
      <td width="131" height="30">现在的缓冲区大小为: </td>
      <td width="223"><%=response.getBufferSize()%></td>
    </tr>
  </table>
</body>
</html>

```

程序运行结果如图 5.11 所示。



图 5.11 显示缓冲区的使用情况

5.3.4 Response 方法一览

Response 对象的常用方法如表 5.2 所示。

表 5.2 Response 对象的常用方法

方法名称	含 义
addHeader(String name,String value)	添加 HTTP 头文件, 该 Header 将会传到客户端去, 如果有同名的 Header 存在, 那么原来的 Header 会被覆盖
setHeader(String name,String value)	使用给定的名称设置一个 HTTP 文件头的值, 如果该值存在, 那么它将会被新的值覆盖
containsHeader(String name)	判断指定名字的 HTTP 文件头是否存在, 并返回布尔值
flushBuffer()	强制将当前缓冲区的内容发送到客户端
addCookie(Cookie cookie)	添加一个 Cookie 对象, 用来保存客户端的用户信息, 可以用 request 对象的 getCookies() 方法获得这个 Cookie
sendError(int sc)	向客户端发送错误信息。例如: “505 指示服务器内部错误”, “404 指示网页找不到的错误”
sendRedirect(URL)	把响应发送到另一个指定的页面 (URL) 进行处理
setLocale(java.util.Locale loc)	设置本地的国家和语言
getCharacterEncoding()	获取字符编码方式
getOutputStream()	获取到客户端的输出流对象
setContentType(String type)	设置响应的 MIME 类型

5.4 会话对象: Session

Session 是一种服务器单独处理与记录用户端使用者信息的技术。当使用者与服务器联机时, 服务器给每个使用者一个 Session, 并设定其中的内容。这些 Session 是相互独立的, 服务器端可以借此来辨别不同的使用者并分别提供服务。

5.4.1 Session 原理

Session 对象是 javax.servlet.http.HttpSession 类的子类对象, 它封装了属于客户会话的所有信息。

当一个用户首次访问服务器上某个 JSP 页时, JSP 引擎会产生一个 Session 对象, 同时分配一个 String 类型的 ID, 并将这个 ID 发往客户端, 存放于 Cookie 中, 这样 Session 对象和用户之间就建立起一一对应的关系。

系统自动分配给用户的 Session 标志可以通过 getId() 方法得到, 如下:

```
<body>
    客户端 Session 的 ID 值: <%=session.getId()%>
</body>
```

Session 中的 ID 标志是唯一的, 用来标识每个用户, 上段代码的运行结果如图 5.12 所示。

客户端Session的ID值: 80FA84085083A4D0F2D77022767C5B02

图 5.12 获取客户端 Session 标志

注意: 在实际运行时获得的 ID 不一定与图 5.12 中的一样, 但就一次会话来说, 这个值是唯一的, 刷新浏览器时, 该标志的值不变。

当用户再次访问连接该服务器的其他页面时, 就不再分配新的 Session 对象, 直到关闭浏览器时, 服务器端该用户的 Session 对象才被取消, 并且和用户的对应关系也取消。若重新打开浏览器再连接到该服务器时, 服务器会为用户创建一个全新的 Session 对象。

5.4.2 数据存取

1. 设置与获取属性

Session 内置对象可以使用 `setAttribute()` 方法保存属性的名称和值, 如果程序员想要获得保存在 Session 中的信息, 则需要调用 `getAttribute()` 方法进行获取。

例如, 设置 String 类型的字符串, 将该字符串保存在 Session 对象内, 之后再从 Session 对象中取出, 并赋值给新的 String 类型的字符串, 代码如下:

```
<%  
    String name1="水漂奇鳧";  
    session.setAttribute("name",name1);           //将 name 属性存储在 session 对象中  
    String name2=(String)session.getAttribute("name");  
%>
```

除了上述方法可以管理 session 中的属性外, 还可以通过 `getAttributeNames()` 方法进行获取, 该方法的语法如下:

```
java.util Enumeration getAttributeNames()
```

该方法返回一个枚举类型的对象, 其中包含绑定在该 session 中所有属性的名称。

【例 5.7】 用 `setAttribute()/getAttribute()` 统计访问页面的人数。

输入以下内容, 以 `5_7sessAttr.jsp` 作为文件名保存:

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>  
<html>  
<head>  
    <title> session 对象的应用 1</title>  
</head>  
<body>  
    <h2>存取 session 数据的实例</h2>  
    <hr><br>  
    <%  
        int num=1; //num 记录来访人数  
        Object obj=session.getAttribute("num");  
        if(obj==null)  
        {  
            //设定 Session 对象的 num 变量的值  
            session.setAttribute("num",String.valueOf(num));  
        }  
    %>
```

```
else
{
    //取得 Session 对象中的 num 变量
    num=Integer.parseInt(obj.toString());
    num+=1;        //来访人数增加 1 个
    session.setAttribute("num",String.valueOf(num));
}
%>
页面访问的人数为:
<%= num %>
</body>
</html>
```

第 1 次运行结果如图 5.13 所示。

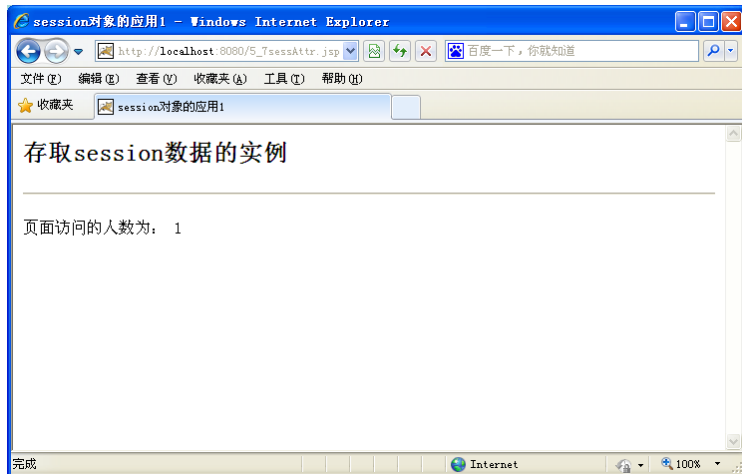


图 5.13 第 1 次运行

将该页面连续刷新 9 次（相当于第 10 次被访问）后的运行结果如图 5.14 所示。

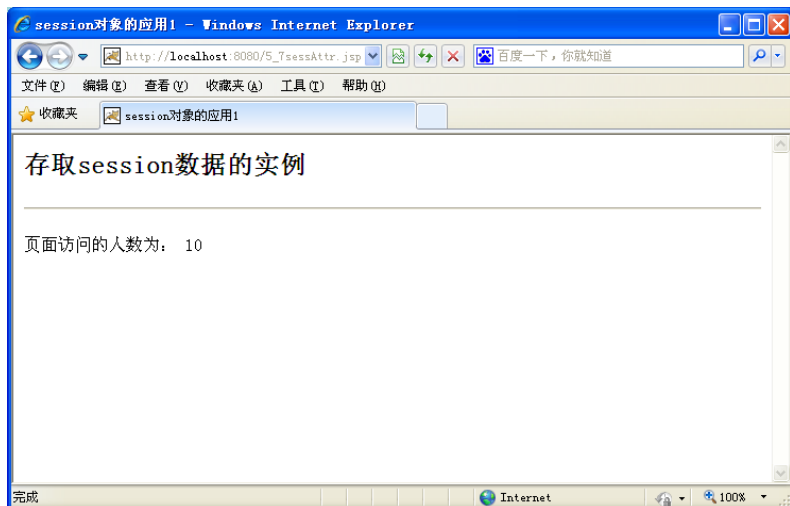


图 5.14 第 10 次被访问

2. 移除属性

在 JSP 页面中可以将已经保存到 Session 中的任何属性进行移除操作。Session 对象使用 `removeAttribute()` 方法所提供的名称实现移除功能，从一个 Session 中删除指定的绑定属性，语法如下：

```
void removeAttribute(java.lang.String name)
```

参数 `name` 为 `String` 类型的值，代表要移除的属性的名称。

在 JSP 页面中，还可以通过 Session 对象中的 `invalidate()` 方法删除已经保存到该 Session 中的所有属性。`invalidate()` 方法的语法如下：

```
void invalidate()
```

5.4.3 超时管理

Session 是用于保存客户端信息而分配给客户端的对象，但 HTTP 协议不能保存客户端请求信息的历史记录，为了解决这一问题，一旦生成了一个 Session 对象，服务器和客户端之间的连接就会一直保持下去。但是，如果在一定时间（系统默认为 30 分钟）内，客户端不向服务器发请求，Session 对象还是会自动消失的。

在实际应用中，30 分钟的有效时间对某些网站来说有些短，但对另一些网站来说又显得长了。因此，为了减少服务器资源的浪费，就应该使用超时设置以确定客户端 session 是否存在。因为 Web 客户端在进入非活动状态时不通知服务器，为了清除存储在 Session 对象中的客户申请资源，Servlet 容器设置了一个超时窗口。在非活动的时间超出窗口的超时大小时，JSP 容器将使 Session 对象无效并撤销所有属性的绑定，从而管理 session 的生命周期。

Session 对象用于超时管理的方法有下面三个。

- `getLastAccessedTime()`：获取客户端最近访问服务器端的保存时间。
- `getMaxInactiveInterval()`：获取客户端停止访问服务器端的保存时间。
- `setMaxInactiveInterval(int value)`：设置客户端停止访问后，session 在服务器端的保存时间。

例如，设置有效时间为 200 秒，如下：

```
<%session.setMaxInactiveInterval(200);%>
```

5.4.4 Session 方法及应用

1. Session 方法一览

Session 对象的常用方法如表 5.3 所示。

表 5.3 Session 对象的常用方法

方法名称	含义
<code>getAttribute(String name)</code>	获得指定名字的属性，如果该属性不存在，将会返回 <code>null</code>
<code>getAttributeNames()</code>	返回 Session 对象中存储的每个属性对象，结果集是一个 <code>Enumeration</code> 类的实例
<code>getCreationTime()</code>	返回 Session 对象被创建的时间，单位为毫秒
<code>getId()</code>	返回 Session 对象在服务器端的编号。每生成一个 Session 对象，服务器都会给它一个编号，而且这个编号不会重复，这样服务器才能根据编号来识别 Session，并且正确地处理某一特定的 Session 及其提供的服务

续表

方法名称	含 义
getLastAccessedTime()	返回当前 Session 对象最后一次被操作的时间，单位为毫秒
getMaxInactiveInterval ()	获取 Session 对象的生存时间，单位为秒
setMaxInactiveInterval (int interval)	设置 Session 对象的有效时间（超时时间），单位为秒
removeAttribute(String name)	删除指定属性的属性值和属性名
setAttribute(String name,Java.lang.Object value)	设定指定名字的属性，并且把它存储在 Session 对象中
invalidate()	注销当前的 Session 对象

2. Session 方法综合应用

下面通过一个例子演示一下表 5.3 中各个方法的应用。

【例 5.8】Session 对象主要方法的应用。

输入以下内容，以 5_8sessMethod.jsp 作为文件名保存：

```
<%@ page language="java" contentType="text/html; charset=gb2312" %>
<html>
<head>
    <title>session 对象的应用 2</title>
</head>
<body>
    <%
        String username="南京师范大学";
        String password="123456";
        session.setAttribute("username",username);
        session.setAttribute("password",password);
    %>
    <a href="5_8next1.jsp">Session 对象的各种方法的应用</a>
</body>
</html>
```

以下是 5_8next1.jsp 文件的代码，这个页面主要输出各个系统的信息：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="java.util.*" %>
<html>
<head>
    <title> session 转向页面 1</title>
</head>
<body>
    <%
        String usr=(String)session.getAttribute("username");
        String pwd=(String)session.getAttribute("password");
    %>
    用户： <%=usr%><br>
    密码： <%=pwd%><br>
    session create:<br>
    <%out.println(session.getCreationTime());%><br>
```



```

session id:<br>
<%out.println(session.getId());%><br>
session last access:<br>
<%out.println(session.getLastAccessedTime());%><br>
session 原来最大休眠时间为:<br>
<%out.println(session.getMaxInactiveInterval()+"ms");%><br>
session 设置后的最大休眠时间为:<br>
<%session.setMaxInactiveInterval(60*10);%>
<%out.println(session.getMaxInactiveInterval()+"ms");%><br>
<%
    Enumeration e=session.getAttributeNames();
    out.println("-----"<br>");
    while(e.hasMoreElements())
    {
        String name=(String)e.nextElement();
        String value=(String)session.getAttribute(name);
        out.print(name+":");
        out.print(value+"<br>");
    }
%>
<%
    session.removeValue("username");
%>
<br>
<a href="5_8next2.jsp">调用 removeValue()方法</a>
</body>
</html>

```

以下是 5_8next2.jsp 文件的代码，显示指定的用户名和密码：

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title> session 转向页面 2</title>
</head>
<body>
    <%
        String usr=(String)session.getAttribute("username");
        String pwd=(String)session.getAttribute("password");
    %>
    用户: <%=usr%><br>
    密码: <%=pwd%>
</body>
</html>

```

运行程序，初始页面如图 5.15 所示。

单击页面上的链接，转到如图 5.16 所示的页面，显示了 Session 对象几个主要方法的运行结果。



图 5.15 初始页



图 5.16 Session 几个主要方法的运行结果

单击页面下方的链接“调用 removeValue()方法”，由前面的代码可知，该方法删去了 username 属性的值，程序会跳转到如图 5.17 所示的页面。

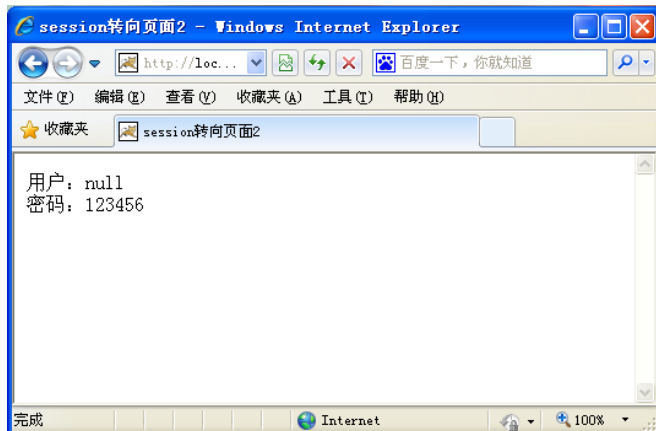


图 5.17 username 属性值已被删除

从图 5.17 的页面上可见, 由于 username 属性的值已被成功删除, 故用户名显示为 null(空)。

5.5 共享对象: Application

Application 对象为多个应用程序保存信息。在 JSP 服务器运行时, 仅有一个 Application 对象, 它由服务器创建, 也由服务器自动清除, 不能被用户创建和删除。

5.5.1 作用范围

Application 对象用于保存所有应用系统中的公共数据, Web 服务器启动并自动创建 Application 对象后, 只要没有关闭服务器, Application 对象就一直存在, 所有用户都可以共享它。

Application 对象与 Session 对象有一定区别, Session 对象和客户端有关, 不同客户端的 Session 是完全不同的对象, 而 Application 对象都是相同的一个对象, 即共享这个内置的 Application 对象。

在 JSP 页面中, 涉及作用范围的对象有 Request、Session 和 Application, 它们之间的关系如图 5.18 所示。

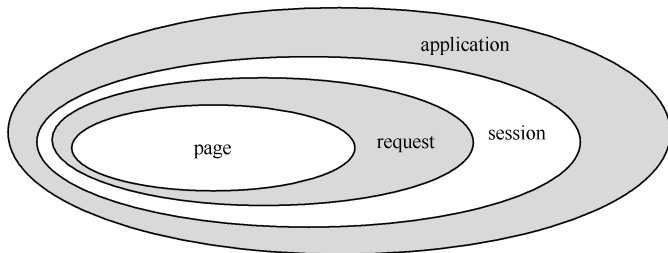


图 5.18 JSP 页面中对象的作用范围

5.5.2 全局网页应用

Application 对象唯一、共享的特性, 使得它非常适合用于网页的全局控制。

【例 5.9】用 Application 对象实现一个网页计数器。

输入以下内容, 以 5_9app1.jsp 作为文件名保存:

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title>application 对象的应用 1</title>
</head>
<body>
    <%
        int num=1;
        Object obj=null;
        obj=application.getAttribute("num");
```

```
        if(obj==null)
        {
            application.setAttribute("num",String.valueOf(num));
        }
        else
        {
            num=Integer.parseInt(obj.toString())+1;
            application.setAttribute("num",String.valueOf(num));
        }
    }
    %>
    <font color="blue">页面访问的人数为: </font>
    <font color="red"><%= num %></font><br>
</body>
</html>
```

该页面第 10 次被访问后的运行结果如图 5.19 所示。



图 5.19 第 10 次被访问

本例与之前的【例 5.7】的不同之处在于：当关掉浏览器窗口，然后重新打开另一个窗口运行本程序时，计数值会在已有的基础上持续增加而不是从 1 开始重新计数，故 Application 对象在网站设计中常常用来统计站点的总体访问量。若要使本例的计数重新回归到 1，则必须重启 Tomcat 服务器。

【例 5.10】用 Application 对象获得各种系统信息。

输入以下内容，以 5_10app2.jsp 作为文件名保存：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="java.util.*" %>
<html>
<head>
    <title>application 对象的应用 2</title>
</head>
<body>
    <%
        String username="南京师范大学";
```

```

String password="123456";
application.setAttribute("username",username);
application.setAttribute("password",password);
Enumeration enumer=application.getAttributeNames();
while(enumer.hasMoreElements())
{
    String name=(String)enumer.nextElement();
    out.println(name+"----"+application.getAttribute(name)+"<br>");
}
%>
</body>
</html>

```

运行结果如图 5.20 所示。

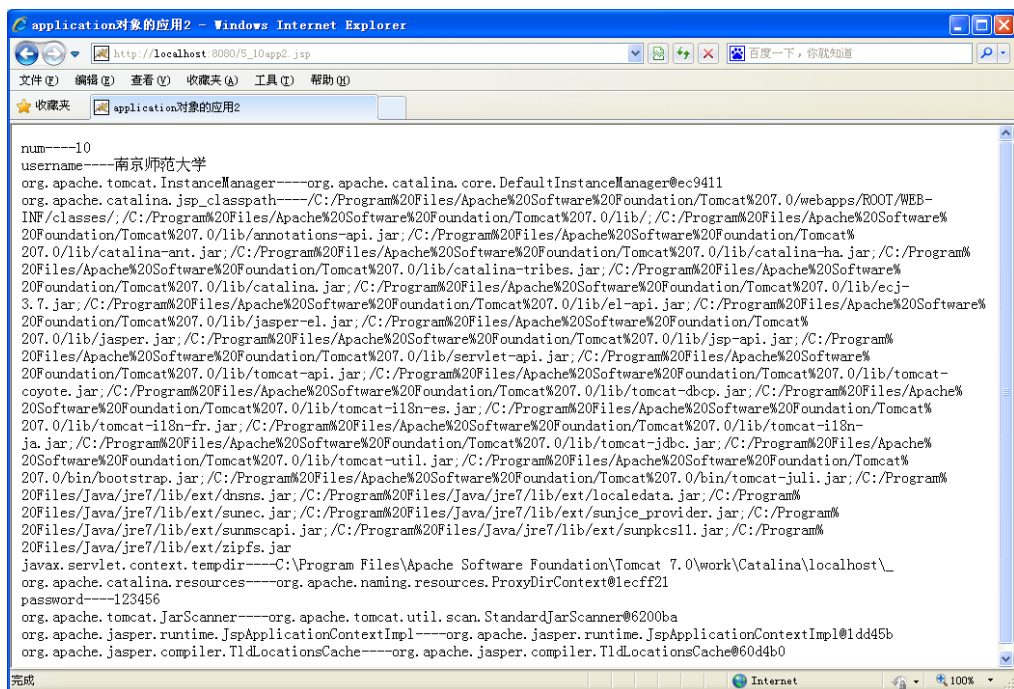


图 5.20 结果显示

5.5.3 Application 方法一览

Application 对象的常用方法如表 5.4 所示。

表 5.4 Application 对象的常用方法

方法名称	含义
getAttribute(String name)	返回由 name 名字指定的 Application 对象的属性的值。返回值是一个 Object 对象，如果没有，则返回 null
getAttributeNames()	返回所有 Application 对象属性的名字，结果集是一个 Enumeration 类的实例
getInitParameter(String name)	返回由 name 名字指定的 Application 对象的某个属性的初始值，如果没有参数，就返回 null

续表

方法名称	含 义
getServerInfo()	返回 Servlet 编译器当前版本的信息
setAttribute(String name, Object obj)	将参数 Object 指定的对象 obj 添加到 Application 对象中，并为添加的对象指定一个属性
removeAttribute(String name)	删除一个指定的属性

5.6 其他对象

之前的 5.2~5.5 节分别详细地介绍了 JSP 四个最重要的服务器对象，本节介绍余下的五个对象。

5.6.1 Out 对象

Out 对象是指在服务器中向客户端打开的 Output Stream。JSP 可以利用 Out 对象，把除脚本小程序以外的所有信息发送到客户端的浏览器中，并且 Out 对象还管理应用服务器上的输出缓冲区，其基类为 JspWriter。

Out 对象的主要方法如表 5.5 所示。

表 5.5 Out 对象的主要方法

方法名称	含 义
print()/println()	输出各种类型的数据
clearBuffer()	清除缓冲区的数据，并将数据写入客户端
clear()	清除缓冲区的当前内容，但不将数据写入客户端
flush()	输出缓冲区中的数据，同时清除缓冲区中的数据
newLine()	输出一个换行符号
getBufferSize()	获取缓冲区的大小，缓冲区大小可以用<%@page buffer="size"%>语句来设置
getRemaining()	获取缓冲区剩余空间的大小
isAutoFlush()	返回布尔值，如果自动缓冲，则返回 true，否则返回 false。是否自动缓冲可以用<%@page isAutoFlush(="true/false" %>语句来设置
close()	关闭输出流

【例 5.11】 Out 对象主要方法的应用。

输入以下内容，以 5_11out.jsp 作为文件名保存：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title> out 对象 应用实例 </title>
</head>
<body>
    <%
        out.println("<b>out 对象应用实例: </b><br><hr>");
        out.println("<br>out.println(boolean):");
        out.println(true);
    %>
```

```

out.println("<br>out.println(char):");
out.println('a');
out.println("<br>out.println(char[]):");
out.println(new char[] {'a','b','c'});
out.println("<br>out.println(double):");
out.println(2.88d);
out.println("<br>out.println(float):");
out.println(16.66f);
out.println("<br>out.println(int):");
out.println(60);
out.println("<br>out.println(long):");
out.println(123456789123L);
out.println("<br>out.println(object):");
out.println(new java.util.Date().toLocaleString());
out.println("<br>out.println(string):");
out.println("hello jsp");
out.println("<br>out.newLine():");
out.newLine();
out.println("<br>out.getBufferSize():");
out.println(out.getBufferSize());
out.println("<br>out.getRemaining():");
out.println(out.getRemaining());
out.println("<br>out.isAutoFlush():");
out.println(out.isAutoFlush());
out.flush();
out.println("<br>调用 out.flush()");
out.close();
out.println(2.88d);
out.println(60);

%>
</body>
</html>

```

运行结果如图 5.21 所示。

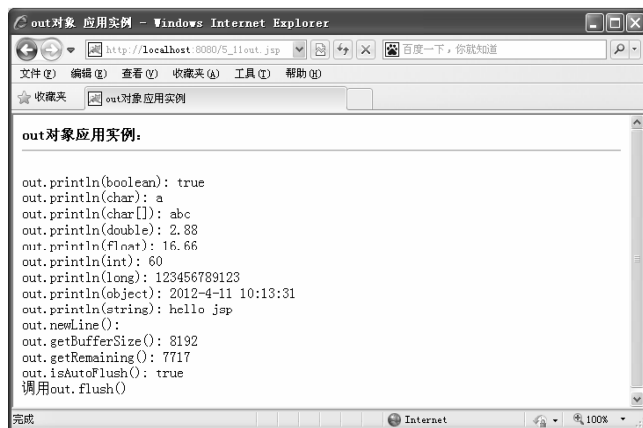


图 5.21 Out 对象的应用

注意：out.close()之后的 out.println(2.88d)和 out.println(60)并没有被显示出来。

5.6.2 Page 对象

Page 对象的实质就是 java.lang.Object，它是 java.lang.Object 类的一个实例。Page 对象代表 JSP 本身，也就是说，它是代表 JSP 转换后的 Servlet 的，可以调用 Servlet 类的方法，其作用与 Java 中的 this 相同。

Page 对象的常用方法如表 5.6 所示。

表 5.6 Page 对象的常用方法

方法名称	含 义
getClass()	获取 page 对象的类
hashCode()	获取 page 对象的 hash 码
equals(Object obj)	判断 page 对象是否与参数中的 obj 相等
copy(Object obj)	把此 page 对象复制到指定的 Object 对象中
clone()	克隆当前的 page 对象
toString()	把 page 对象转换成 String 类型的对象

使用 Page 输出 JSP 页面的 ID 和 hash 代码值的具体实现代码如下：

```
<%  
    int hashCode=page.hashCode();  
    String thisStr=page.toString();  
    out.println("page 对象的 ID 值: "+thisStr);  
    out.print("<br>");  
    out.println("page 对象的 hash 代码"+hashCode);  
%>
```

上段代码运行结果如图 5.22 所示。

```
page对象的ID值: org.apache.jsp.pageID_jsp@24ea0e  
page对象的hash代码2419214
```

图 5.22 Page 输出 JSP 页面的 ID 和 hash 代码值

5.6.3 PageContext 对象

PageContext 对象是一个比较特殊的对象，它的作用是取得任何范围的参数，通过它可以获取 JSP 页面的 Out、Request、Response、Session、Application 等对象，或者重新定向客户端的请求等。PageContext 的创建和初始化都是由容器来完成的，在 JSP 页面里可以直接使用其句柄，它的 getXXX()、setXXX()和 findXXX()方法可以用来根据不同的对象作用范围实现对这些对象作用域的管理。

PageContext 对象的常用方法如表 5.7 所示。

表 5.7 PageContext 对象的常用方法

方法名称	含 义
setAttribute(String name,Object attribute)	设置默认页面范围或特定对象范围之中的已命名对象属性
getAttribute(String name [, int scope])	获取一个已命名为 name 的对象的属性，可选参数 scope 表示在特定范围内
removeAttribute(String name,[int scopel)	删除指定范围内的某个属性
forward(String relativeUrlPath)	将当前页面重定向到其他页面
include(String relativeUrlPath)	在当前位置包含另一个文件
release()	释放 pageContext 对象所占用的资源
getServletContext()	获取当前页的 ServletContext 对象
getException()	获取当前页的 Exception 对象

【例 5.12】PageContext 对象获取作用域的值。

输入以下内容，以 5_12pageContext.jsp 作为文件名保存：

```
<%@ page language="java"    contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title> pageContext 对象 应用实例 </title>
</head>
<body>
    <%
        request.setAttribute("zaq","获取到 request 范围内对象的值");
        session.setAttribute("zaq","获取到 session 范围内对象的值");
        application.setAttribute("zaq","获取到 application 范围内对象的值");
    %>
    <table border="1">
        <tr>
            <td width="138">request 范围内: </td>
            <td width="251"><%=pageContext.getRequest().getAttribute("zaq")%></td>
        </tr>
        <tr>
            <td>session 范围内: </td>
            <td><%=pageContext.getSession().getAttribute("zaq")%></td>
        </tr>
        <tr>
            <td>application 范围内: </td>
            <td><%=pageContext.getServletContext().getAttribute("zaq")%></td>
        </tr>
    </table>
</body>
</html>
```

程序运行结果如图 5.23 所示。



图 5.23 PageContext 对象获取作用域的值

5.6.4 Config 对象

Config 对象的主要作用是取得服务器的配置信息。Config 对象为 `javax.servlet.Config` 接口的类实例对象，通过 `pageContext.getServletConfig()` 方法可以获取一个 Config 对象。开发者可以为应用程序环境在 `web.xml` 文件中为 Servlet 程序和 JSP 页面提供初始化参数。

Config 对象的常用方法如表 5.8 所示。

表 5.8 Config 对象的常用方法

方法名称	含义
<code>getServletContext()</code>	获取当前的 Servlet 上下文
<code>getInitParameter(String name)</code>	获取指定的初始参数的值
<code>getInitParameterNames()</code>	获取所有的初始参数的值
<code>getServletName()</code>	获取当前的 Servlet 名称

【例 5.13】 Config 对象获取 `web.xml` 文件中的初始化参数。

① 在 MyEclipse 中创建 Web 项目，取名为 `5_13config`。在 `web.xml` 配置文件中设置 Servlet 初始化参数，代码如下：

```
<servlet>
    <servlet-name>zaq</servlet-name>
    <jsp-file>/index.jsp</jsp-file>
    <init-param>
        <param-name>email</param-name>
        <param-value>easybooks@163.com</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>zaq</servlet-name>
    <url-pattern>/index.jsp</url-pattern>
</servlet-mapping>
```

② 在 index.jsp 页面中通过 Config 对象中的 getInitParameter()方法获取在 web.xml 文件中初始化了的参数，代码如下：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title> config 对象 应用实例 </title>
</head>
<body>
    <center>
        易睿得邮箱地址: <%=config.getInitParameter("email")%>
    </center>
</body>
</html>
```

③ 程序运行结果如图 5.24 所示。



图 5.24 Config 对象获取初始化参数

5.6.5 Exception 对象

Exception 对象用来处理 JSP 文件在执行时所发生的错误和异常。Exception 对象可以配合 page 指令一起使用，通过指定某一页面为错误处理页面，把所有的错误都集中到那个页面进行处理。这样可以使得整个系统更加健壮，也使得程序的流程更加清晰，这也是 JSP 比 ASP 和 PHP 先进的地方。

Exception 对象的常用方法如表 5.9 所示。

表 5.9 Exception 对象的常用方法

方法名称	含 义
getMessage()	获得当前的错误信息
getLocalizedMessage()	本地化语言的异常错误
printStackTrace()	以标准错误的形式输出一个错误和错误的堆栈跟踪
fillInStackTrace()	重写异常的执行栈轨迹
toString()	以字符串的形式返回一个对异常的描述

注意：只有在 isErrorPage=true 的情况下才可以使用 Exception 对象。

【例 5.14】使用 JSP 的异常处理机制，使得页面交互更加富有变化，程序更加强壮。输入以下内容，以 5_14exce.html 作为文件名保存：

```

<html>
<head>
    <title>整数除法</title>
</head>
<body bgcolor="#ffccce">
    <div align="center">
        <form method="post" action="5_14exce.jsp">
            <p>被除数<input type="text" name="value1"></p>
            <p>除 数<input type="text" name="value2"></p>
            <p>
                <input type="submit" name="sub" value="计算">
                <input type="reset" name="res" value="取消">
            </p>
        </form>
    </div>
</body>
</html>

```

以下是 5_14exce.jsp 文件的代码:

```

<%@ page language="java" errorPage="5_14error.jsp" %>
<%@ page contentType="text/html; charset=gbk"%>
<html>
<head>
    <title>整数除法</title>
</head>
<body bgcolor="#ffccce">
    <center>
        <%
            int a = 0;
            int b = 0;
            int c = 0;
            try{
                a = Integer.parseInt( request.getParameter( "value1" ) ); //将字符串转换成整型
            }
            catch( NumberFormatException e ){
                throw new NumberFormatException( "被除数非整数! " );
            } //当出现字符串无法转换为整型的异常时, 抛出异常
            try{
                b = Integer.parseInt( request.getParameter( "value2" ) );
            }
            catch( NumberFormatException e ){
                throw new NumberFormatException( "除数非整数! " );
            }
            c=a/b;
            out.println("<h2>计算结果为: </h2>");
            out.println( a + " / " + b + " = " + c );
        %>
    </center>

```

```

        <br><br><br>
        <a href="javascript: history.back();">返回</a>
    </center>
</body>
</html>

```

如果本程序出现异常, 指定由 5_14error.jsp 页面进行处理, 以下是 5_14error.jsp 文件的代码:

```

<%@ page language="java" isErrorPage="true" contentType="text/html; charset=gbk"%>
<html>
<head>
    <title>异常处理页面</title>
</head>
<body bgcolor="#ffcccc">
    <div align="center"><br>
        <h1>错误信息</h1>
        <hr>
        <p>
            <center>
                <h3><%= exception.toString() %></h3>
                <a href="javascript: history.back();">返回</a>
            </center>
        </p>
    </div>
</body>
</html>

```

为了使浏览器能显示异常信息, 在运行程序之前, 要先进行设置。打开 IE, 单击“工具”→“Internet 选项”→“高级”命令, 选中“显示每个脚本错误的通知”复选框, 取消选中“显示友好 http 错误信息”复选框, 如图 5.25 所示。

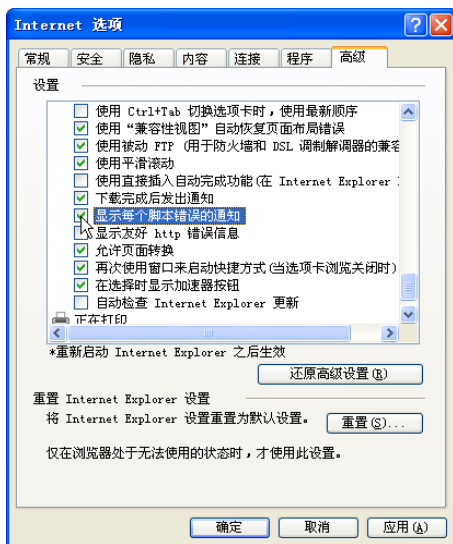


图 5.25 设置 IE

这样访问的时候 IE 浏览器就会提示具体错误信息, 根据错误信息修改代码。

运行 5_14exce.html 文件，显示计算整数除法的页面，如图 5.26 所示。

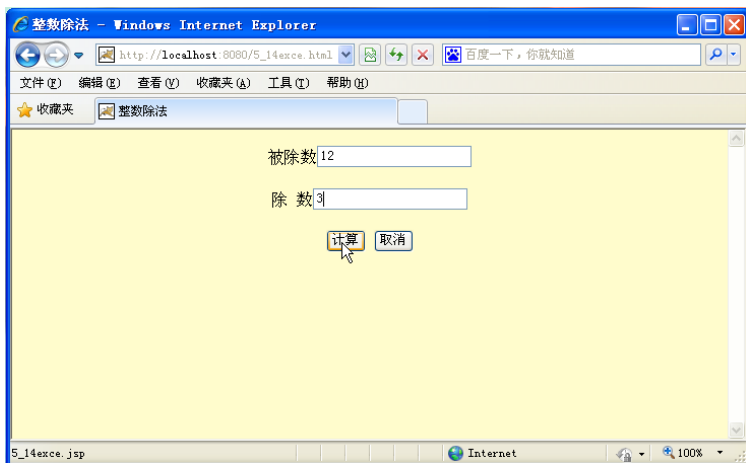


图 5.26 计算整数除法页

① 输入被除数“12”、除数“3”，然后单击“计算”按钮，结果如图 5.27 所示。

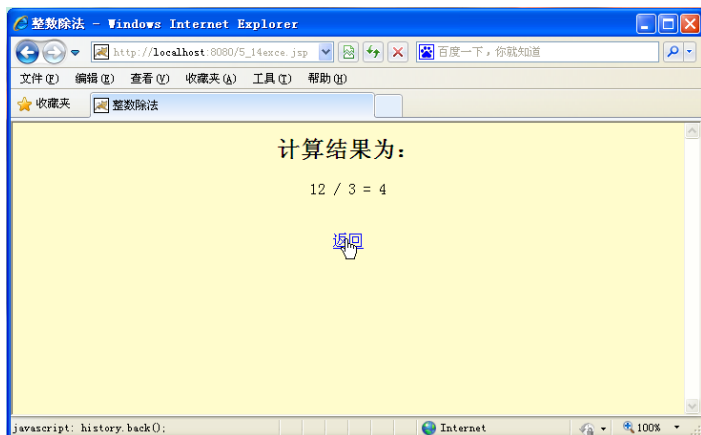


图 5.27 计算结果显示

② 单击“返回”按钮，回到输入界面，重新输入除数“a”，然后单击“计算”按钮，提示错误信息“除数非整数！”，如图 5.28 所示。

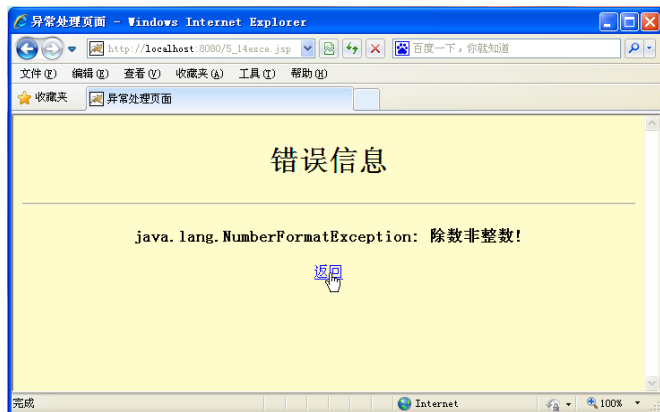


图 5.28 “除数非整数！”错误

③ 单击“返回”按钮，回到输入界面，重新输入除数“0”，然后单击“计算”按钮，结果如图 5.29 所示。

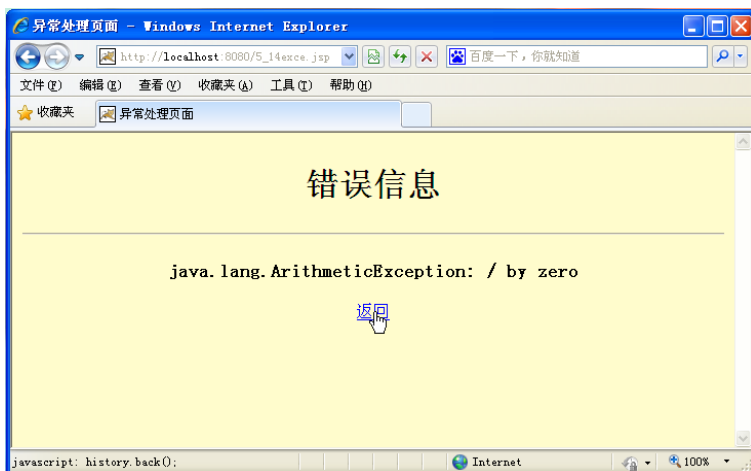


图 5.29 被 0 除出错

5.7 Cookie 及应用

Cookie 是一小段文本信息，由 Web 服务器送到客户端存储。当该客户端再次访问同一 Web 站点时，这些信息会不做任何修改地被送回 Web 服务器，它是一种 Web 服务器通过浏览器在访问者的硬盘上存储信息的手段。

5.7.1 创建 Cookie 对象

创建 Cookie 对象的语法格式如下：

```
Cookie 对象名称 = new Cookie("索引值","数据");
```

Cookie 对象不能单独使用，必须与 Request 对象或 Response 对象结合使用才起作用。将 Cookie 信息传送到客户端的方法如下：

```
response.addCookie(对象名称);
```

例如：

```
Cookie newCookie = new Cookie("username","周何骏");
```

```
response.addCookie(newCookie);
```

取得客户端所有的 Cookie 对象的数组的方法如下：

```
request.getCookie();
```

5.7.2 Cookie 对象的主要方法

Cookie 对象的主要方法如表 5.10 所示。

表 5.10 Cookie 对象的主要方法

方法名称	含 义
getName()	返回 Cookie 的名字。返回值的类型为 String 类型
getValue()	返回 Cookie 的值。返回值的类型为 String 类型
setValue(String newValue)	Cookie 创建后设置一个新的值
setMaxAge(int Age)	以秒计算，设置 Cookie 的存在期限

例如：

```
Cookie user = new Cookie("username","周何骏");
user.setMaxAge(3600);
response.addCookie(user);
```

5.7.3 Cookie 对象与 Session 对象的比较

用 Response 对象可以建立 Cookie 文件，以记录来访客户的各种信息。Session 对象的概念与 Cookie 很相似，也可以用来记录客户的状态信息。

两者不同的是：Cookie 把信息记录在客户端中，而 Session 对象则把信息记录在服务器中。处理速度上，Cookie 对象比 Session 对象要快。

5.7.4 示例

【例 5.15】使用 Cookie 对象在浏览器中暂时保存少量的信息。

输入以下内容，以 5_15cook.jsp 作为文件名保存：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%
    request.setCharacterEncoding("gb2312");
    String name = request.getParameter("Name");    //取得文本框的内容
    String pass = request.getParameter("Pass");
    Cookie cookies[] = request.getCookies();        //取得所有的 Cookie 对象
    if(name != null) {
        Cookie c = new Cookie("Name", name);      //新建 Cookie，命名为 Name
        c.setMaxAge(30);                          //Cookie 的有效期为 30 秒
        response.addCookie(c);                    //实现写入
    }
    else if(cookies != null) {
        //如果已经设置了 cookie，则得到它的值，并保存到变量 name 中
        for(int i=0; i<cookies.length; i++) {
            if(cookies[i].getName().equals("Name"))
                name = cookies[i].getValue();      //取得 cookie 值
        }
    }
    if(pass != null) {
        Cookie c = new Cookie("Pass", pass);
        c.setMaxAge(30);
```



```
        response.addCookie(c);
    }
    else if(cookies != null)  {
        for(int i=0; i<cookies.length; i++) {
            if(cookies[i].getName().equals("Pass"))
                pass = cookies[i].getValue();
        }
    }
}
```

```
<html>  
<head>  
    <title>使用 Cookie 保存信息</title>  
</head>  
<body>  
    <h2 align="left"><font color="#660000">利用 Cookies 对象保存信息</font></h2>  
    <hr>  
    <form action="5_15cook.jsp" method="post">  
        <p>用户名:  
            <input type="text" size="20" name="Name" value=""  
                <% if(name != null) out.println(name); %>">  
        </p>  
        <p>密码:   <br>  
            <input type="text" size="20" name="Pass" value=""  
                <% if(pass != null) out.println(pass); %>">  
        </p>  
        <p>  
            <input type="submit" value="保 存">  
            <input type="reset" value="取 消">  
        </p>  
    </form>  
</body>  
</html>
```

运行结果如图 5.30 所示。

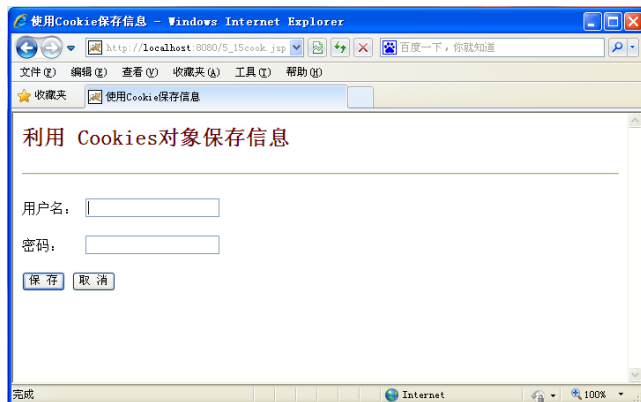


图 5.30 初始页面

在对话框中输入用户名 easybooks 和密码 123456 后,单击“保存”按钮,就将页面输入的信息保存在 Cookie 中,然后关闭浏览器。当再次运行此程序时,将看到保存在 Cookie 中的值,运行结果如图 5.31 所示。

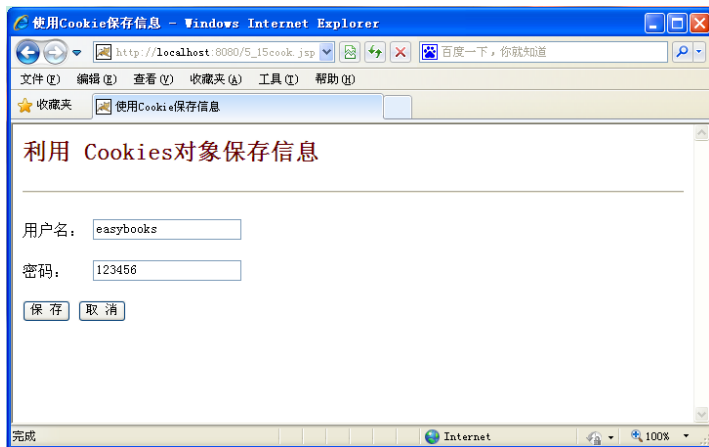


图 5.31 看到保存在 Cookie 中的值

5.8 综合应用——简易留言板

本章最后将通过例子说明服务器对象的综合应用。

【例 5.16】简易留言板的制作。

在浏览器中输入 5_16userLogin.html,系统显示登录界面,输入用户名后单击“确定”按钮,这里用 isValid()函数来验证用户名和密码是否符合要求。验证通过后则转到留言板的留言信息输入页面 5_16messInput.jsp。在 5_16messInput.jsp 页面的顶部显示目前登录过该页面的总人数,在对应的文本框填写完相关信息后,单击“提交留言”按钮,就把信息提交到服务器上了,这里通过文件 5_16saveMess.jsp 来实现信息的提交功能。提交完信息后,如果想查看相关的留言信息,则可以通过单击“查看留言板”按钮来查看,查看功能通过文件 5_16showMess.jsp 来实现。

① 输入以下内容,以 5_16userLogin.html 作为文件名保存:

```
<html>
<head>
  <title>用户登录</title>
</head>
<script language = "javascript">
<!--
  function isValid()
  {
    if(loginform.username.value == "")
    {
      window.alert("您必须完成账号的输入!");
      document.loginform.elements(0).focus();
      return false;
```

```

    }
    if(loginform.password.value == "")
    {
        window.alert("您必须完成密码的输入!");
        document.loginform.elements(1).focus();
        return false;
    }
    loginform.submit();
}
-->
</script>
<body bgcolor="#ffffee">
    <h1><center> 用 户 登 入</center></h1>
    <center>
        <form action="5_16messInput.jsp" method="post" name="loginform">
            用户名: <input name="username" type="text" ><br><br>
            密 码: <input name="password" type="password" ><br><br>
                   &nbsp;   <input name="ok" type="submit" value="确定">
                   <input name="cancel" type="reset" value="取消">
        </form>
    </center>
</body>
</html>

```

② 输入以下内容，以 5_16messInput.jsp 作为文件名保存：

```

<%@ page contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title>简易留言板</title>
</head>
<body bgcolor="#ffffee">
    <%!
        synchronized void countPeople()
        {
            ServletContext application=getServletContext();
            Integer number=(null==application.getAttribute("Count"))?
                                null:(Integer)application.getAttribute("Count");

            if (number==null)
            {
                number=new Integer(1);
                application.setAttribute("Count",number);
            }
            else{
                number=new Integer(number.intValue()+1);
                application.setAttribute("Count",number);
            }
        }
    %>

```

```

%>
<%
    if (session.isNew())
    {
        countPeople();
        Integer myCount=(Integer)application.getAttribute("Count");
        session.setAttribute("MyCount",myCount);
    }
%>
<p><p>您是第
<font color="blue">
    <b>
        <%
            int a=((Integer)session.getAttribute("MyCount")).intValue();
            out.print(a);
        %>
    </b>
</font>个访问本站的客户
<%
    String username=request.getParameter("username");
    Cookie cookies[] = request.getCookies();
    if(username != null)
    {
        Cookie c = new Cookie("Name", username);
        c.setMaxAge(600);
        response.addCookie(c);
    }
    else if(cookies != null)
    {
        for(int i=0; i<cookies.length; i++)
        {
            if(cookies[i].getName().equals("Name"))
                username = cookies[i].getValue();
        }
    }
%>
<form action="5_16saveMess.jsp" method="post" name="form">
    <font color="#660000" size="4" face="宋体">用户名: </font>
    <input type="text" name="Name" value="<%=username %>">
    <p><font color="#660000" size="4" face="宋体">留言标题: </font>
    <input type="text" name="Title" size="30">
    <p><font color="#660000" size="4" face="宋体">留言: </font><br>
    <textarea name="messages" rows="6" cols="50"></textarea><br>
    <input type="submit" value="提交留言" name="submit">
</form>
<form action="5_16showMess.jsp" method="post" name="form1">

```

```

        <input type="submit" value="查看留言板" name="look">
    </form>
</body>
</html>

```

③ 输入以下内容，以 5_16saveMess.jsp 作为文件名保存：

```

<%@ page contentType="text/html; charset=gb2312"%>
<%@ page import="java.util.*" %>
<html>
<body bgcolor="#ffffee">
    <%!
        Vector v=new Vector();                //定义向量
        int i=0;
        ServletContext application;
        synchronized void sendMessage(String s)
        {
            application=getServletContext();
            i++;
            v.add("NO."+i+s);                //添加到向量末尾
            application.setAttribute("Mess",v);
        }
    %>
    <%
        String name=request.getParameter("Name");
        String title=request.getParameter("Title");
        String messages=request.getParameter("messages");
        if (name==null){
            name="guest"+(int)(Math.random()*10000);
        }
        if(title==null){
            title="无标题";
        }
        if(messages==null){
            messages="无信息";
        }
        String time=new Date().toLocaleString();
        String s="#" + name + "#" + title + "#" + time + "#" + messages + "#";
        sendMessage(s);
        out.print("您的信息已经提交！ ");
    %>
    <br><a href="5_16messInput.jsp">返回 </a>
    <a href="5_16showMess.jsp">查看留言板</a>
</body>
</html>

```

④ 输入以下内容，以 5_16showMess.jsp 作为文件名保存：

```

<%@ page contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.*" %>

```

```

<html>
<body bgcolor="#ffffee">
    <table width="700" align="center" border="2">
        <tr align="center">
            <td bgcolor="pink">序号</td>
            <td bgcolor="pink">留言者姓名</td>
            <td bgcolor="pink">留言标题</td>
            <td bgcolor="pink">留言时间</td>
            <td bgcolor="pink">留言内容</td>
        </tr>
        <%
            Vector v=(Vector)application.getAttribute("Mess");
            for(int i=0;i<v.size();i++)
            {
                String message=(String)v.elementAt(i);
                StringTokenizer fenxi=new StringTokenizer(message,"#");
                int num=fenxi.countTokens();
            %>
            <tr align="center">
                <%
                    for(int k=1;k<=num;k++)
                    {
                        String str=fenxi.nextToken();
                        byte a[]=str.getBytes("ISO-8859-1");
                        str=new String(a);
                        if(k<num){
                            out.print("<td bgcolor=\"pink\">"+str+"</td>");
                        }
                        else {
                            out.print("<td><textarea rows=\"3\" cols=\"30\">"
                                +str+"</textarea></td>");
                        }
                    }
                %>
            </tr>
        <% } %>
    </table>
    <br><br>
    <center><a href="5_16messInput.jsp">返回</a></center>
</body>
</html>

```

首先运行 5_16userLogin.html 文件，将显示如图 5.32 所示的页面。

在“用户名”中输入“zhouhejun”，输入密码，单击“确定”按钮，便打开留言信息输入页面，如图 5.33 所示。



图 5.32 登录页面

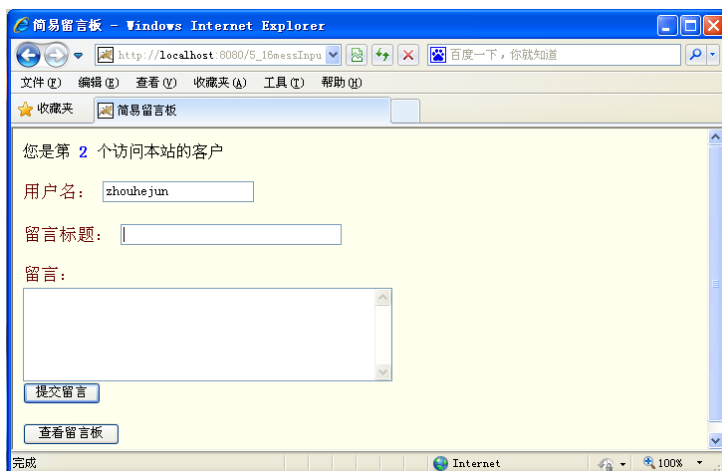


图 5.33 留言信息输入页面

如果你是第 2 位登录该网站的人，便会显示数字“2”；“用户名”中的“zhouhejun”是通过 Cookie 对象获得的，然后输入相应的留言信息，单击“提交留言”按钮后，将显示如图 5.34 所示的画面。



图 5.34 留言提交成功

如果单击“查看留言板”后,将会显示如图 5.35 所示的画面;“用户名”为“zhouhejun”的用户留言信息将出现在留言板中,其中序号为“NO.2”。

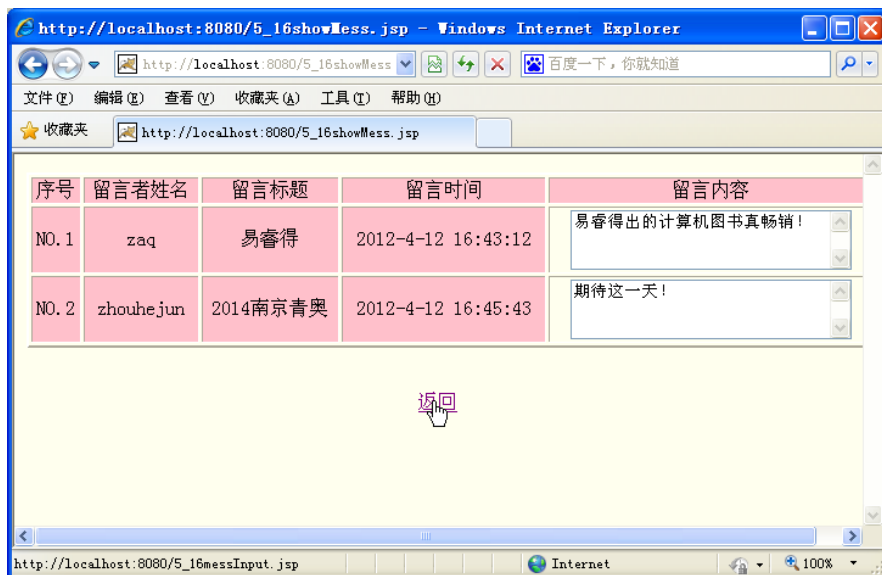


图 5.35 留言板上的内容

代码分析:

由于 `Application` 对象对所有的客户都是相同的,任何客户对该对象中存储的数据的改变都会影响到其他客户,因此,在某些情况下,对该对象的操作需要实现同步处理。

在 `5_16messInput.jsp` 文件中,用 `Application` 对象实现一个计数器,将计数存放在 `Application` 对象中,每个客户对该对象中“计数”的改变都会影响到其他客户。

有些服务器不直接支持使用 `Application` 对象,必须用 `ServletContext` 类声明这个对象。再使用 `getServletContext()` 方法对这个 `Application` 对象进行初始化,实现代码如下:

```
<%!
    synchronized void countPeople()
    {
        ServletContext application=getServletContext();
        Integer number=(null==application.getAttribute("Count"))?null:
            (Integer)application.getAttribute("Count");
        if (number==null)
        {
            number=new Integer(1);
            application.setAttribute("Count",number);
        }
        else{
            number=new Integer(number.intValue()+1);
            application.setAttribute("Count",number);
        }
    }
%>
<%
```



```

        if (session.isNew())
        {
            countPeople();
            Integer myCount=(Integer)application.getAttribute("Count");
            session.setAttribute("MyCount",myCount);
        }
    %>
    <p><p>您是第<font color="blue"><b>
    <%
        int a=((Integer)session.getAttribute("MyCount")).intValue();
        out.print(a);
    %>
    </b></font>>个访问本站的客户

```

在本例中，客户通过 5_16messInput.jsp 向 5_16saveMess.jsp 页面提交用户名、留言标题和留言内容，5_16saveMess.jsp 页面获取这些内容后，用同步方法将这些内容添加到一个向量中，然后再将这个向量添加到 Application 对象中，实现代码如下：

```

ServletContext  application;
synchronized void sendMessage(String s)
{
    application=getServletContext();
    i++;
    v.add("NO."+i+s);                //添加到向量末尾
    application.setAttribute("Mess",v);
}

```

当用户单击“查看留言板”时，5_16showMess.jsp 负责显示所有客户的留言内容，即从 Application 对象中取出向量，然后用字符串分割函数 StringTokenizer() 分离出编号、用户名、留言标题、留言时间和留言内容，最后用表格的方式显示出来，实现代码如下：

```

Vector v=(Vector)application.getAttribute("Mess");           //从 Application 中取出所有内容
String message=(String)v.elementAt(i);                       //取出一条留言
StringTokenizer fenxi=new StringTokenizer(message,"#");      //分离各项

```

在这里使用了向量这种数据结构，Java 的 java.util 包中的 Vector 类负责创建一个向量对象。当创建一个向量时，不用像数组那样必须要给出数组的大小。

向量创建后，例如：Vector a=new Vector(); a 可以使用 add(Object obj) 方法把任何对象添加到向量的末尾，向量的大小会自动增加；可以使用 add(int index,Object obj) 把一个向量添加到该向量的指定位置；可以使用 elementAt(int index) 获取指定索引处的向量的元素（索引初始位置是 0）；可以使用 size() 方法获取向量所含有的元素的个数。

需要注意的是，虽然可以把任何一种 Java 的对象放入一个向量，但是，当从向量中取出一个元素时，必须使用强制类型转换运算符，将其转化为原来的类型。

5.9 上机练习

1. 设计网页 5_4rspHeader.jsp，其显示如图 5.36 所示（与图 5.7 相同）。

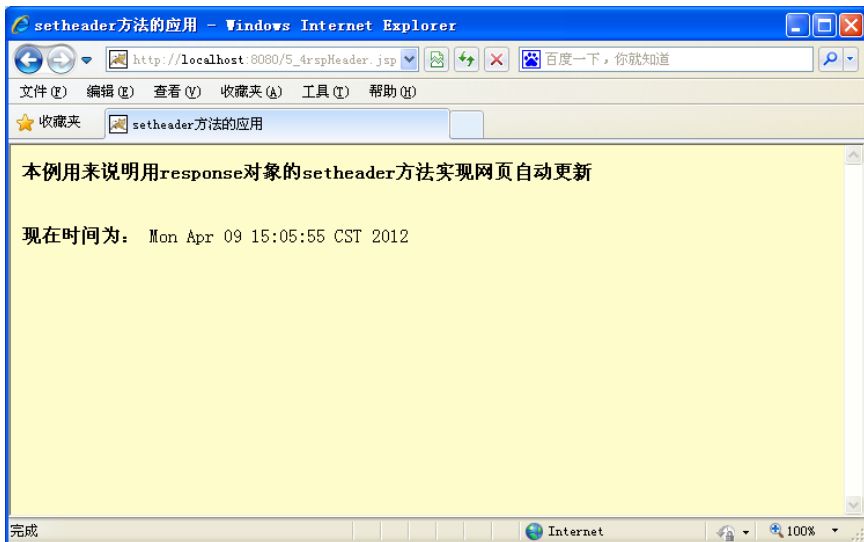


图 5.36 利用 HTTP 文件头自动更新网页

(1) 修改代码, 将页面每隔 5 秒刷新一次改为每隔 10 秒刷新一次; 修改时间的显示方式, 以年、月、日、时、分、秒格式显示。

(2) 修改代码, 利用 Response 对象的 setContentType 方法, 实现如下功能: 当一个用户访问一个 JSP 页面时, 实现动态改变 ContentType 属性值来响应用户。

2. 按照 5.8 节的指导完成简易留言板程序, 对照本书测试其功能, 并完成以下的扩展修改。

(1) 修改 5_16messInput.jsp 文件, 把客户计数器去掉, 改用多框架形式显示在网页的顶部。

(2) 修改 5_16showMess.jsp 文件, 不要用表格形式显示留言板的信息, 而改用列表形式显示每个用户的留言信息。

JavaBean 及其应用

JavaBean 是使用 Java 语言描述的软件组件模型，简单地说，它就是一个可以重复使用的 Java 类。JavaBean 可分为可视化组件和非可视化组件，其中可视化组件包括简单的 GUI 元素（如文本框、按钮）及一些报表组件等。非可视化组件就是没有 GUI 图形用户界面的 JavaBean，主要用来封装业务逻辑（功能）、数据库操作（如数据处理、连接数据库）等。

6.1 JavaBean 简介

6.1.1 使用 JavaBean 的原因

在 JSP 程序中使用 JavaBean 具有以下优点。

1. 实现代码的重复利用

在实际的开发过程中，出现重复的代码或程序段是在所难免的，这会大大降低程序的可重用性并且浪费时间。使用 JavaBean 技术，这个问题就会迎刃而解：作为一种基于 Java 的软件组件，JavaBean 被设计成可以在不同的环境里重复使用。JSP 对于在 Web 应用中集成 JavaBean 组件提供了完善的支持，可以直接利用经测试的可信任的已有组件，避免重复开发。

2. 降低网站系统的耦合度

一个网站系统一般分为数据层、商务层和应用层。若使用 Java 语言环境，可以采用 Servlet 和 JSP 编程。使用纯脚本的 JSP 语言时，如果出现大量用户访问，很快就会达到功能上限。另外，纯脚本语言将应用层和商务层混在一起，耦合度高，修改极不方便。

采用 JavaBean 就可以解决这些问题：可视化的 JavaBean 可以实现控制逻辑与表示层之间的分离，从而大大降低了它们之间的耦合度，而非可视化的 JavaBean 在 JSP 程序中常用来封装事务逻辑、数据库操作等，可很好地实现业务逻辑和前台程序（如 JSP 文件）的分离，使得系统具有更好的健壮性和灵活性。

3. 便于扩充系统功能

JavaBean 组件可以用来执行复杂的计算任务，或负责与数据库的交互以及数据的提取等。作为一种 Java 类，它可以通过封装成为具有某种特定功能或者处理某个特定业务的对象。因此在 JSP+JavaBean 模型中，通过 JavaBean 可以无限扩充 Java 程序的功能，也可以通过 JavaBean 快速生成新的应用程序。

4. 程序容易编写

JavaBean 是 Java 程序的一种,所使用的语法与普通的 Java 程序一致,完全是用 Java 语言编写的,可以在安装了 Java 运行环境的平台上使用,而不需要重新编译。JavaBean 代码不多,但可以被其他程序引用,可以压缩在 jar 文件中,以更小的体积在网络中应用。

5. 系统维护方便

举一个简单的例子,比如一个购物车程序,要实现向购物车中添加一件商品这样的功能,可以写一个购物车操作的 JavaBean,建立一个 public 的 addItem 成员方法,在前台 JSP 文件中直接调用这个方法来实现。如果以后考虑到添加商品时需要先判断库存是否有货,没有货不得购买,此时就可以直接修改 JavaBean 的 addItem()方法,而完全不用修改前台的 JSP 程序了。若把这些处理操作完全写在 JSP 程序中,页面的代码会很多,修改也麻烦。可见,通过 JavaBean 能很好地实现逻辑封装,系统也容易维护。

6.1.2 JavaBean 的形式和要素

编写 JavaBean 就是编写一个 Java 类,只要会写类就能写出一个 Bean,这个类创建的一个对象称为一个 Bean。为了能让使用这个 Bean 的应用程序构建工具(如 JSP 引擎)知道这个 Bean 的属性和方法,只需在类的方法命名上遵守以下规则。

① 如果类的成员变量的名字是 XXX,那么为了更改或获取成员变量的值,即更改或获取属性,在类中可以使用两种方法。

- getXXX(): 用来获取属性 XXX。

- setXXX(): 用来修改属性 XXX。

② 对于 boolean 类型的成员变量,即布尔逻辑类型的属性,允许使用 is 代替上面的 get 和 set。

③ 类中方法的访问属性都必须是 public 的。

④ 类中如果有构造方法,那么这个构造方法也是 public 的并且无参数。

下面通过一个简单的示例来说明 JavaBean 的形式与要素,具体代码如下:

```
import java.io.Serializable;
public class JavaBeanDemo implements Serializable{           //实现了 Serializable 接口
    JavaBeanDemo(){                                           //无参的构造方法
        private int id;                                       //私有属性 id
        private String name;                                  //私有属性 name
        private int age;                                       //私有属性 age
        private String sex;                                    //私有属性 sex
        private String address;                                //私有属性 address
        public String getAddress(){                             //get()方法
            return address;
        }
        public void setAddress (String address){              //set()方法
            this.address=address;
        }
        public int getAge(){
            return age;
        }
    }
}
```

```
    }  
    public void setAge(int age){  
        this.age=age;  
    }  
    public int getId(){  
        return Id;  
    }  
    public void setId(int id){                                //set()方法  
        Id = id;  
    }  
    public String getName(){                                  //get()方法  
        return name;  
    }  
    public void setName (String name){  
        this.name = name;  
    }  
    public String getSex(){  
        return sex;  
    }  
    public void setSex(String sex){  
        this.sex = sex;  
    }  
}
```

代码说明：

该程序具备了 **JavaBean** 的所有要素及形式。声明了五个私有属性并且为这五个属性分别提供了 **setXXX()**与 **getXXX()**方法。

6.2 JavaBean 基本结构

JavaBean 的基本结构分为属性、方法和事件三个部分。

6.2.1 JavaBean 的属性

JavaBean 的属性用于描述 **JavaBean** 的状态，如颜色、大小等，与普通的 **Java** 程序中的属性在概念上非常相似。在 **JavaBean** 设计中，按照属性的不同作用又可以细分为四类，分别是简单（**Simple**）属性、索引（**Indexed**）属性、束缚（**Bound**）属性、限制（**Constrained**）属性。本小节将详细介绍这四类属性的相关知识。

1. Simple 属性

一个 **Simple** 属性表示一个伴随有一对 **get/set** 方法的变量。属性名与该属性相关的 **get/set** 方法名对应。例如，如果有 **setX()**和 **getX()**方法，则暗指有一个名为“**X**”的属性。如果有一个方法名为 **isX**，则通常暗指“**X**”是一个布尔属性（即 **X** 的值为 **true** 或 **false**）。

Simple 属性的用法如下：

```
public class example1 extends Canvas
{
    //属性名为 outString，类型为字符串
    private String outString = "Hello";
    //example1()是 example1 的构造函数，与 C++中构造函数的意义相同
    public example1()
    {
        setBackground(Color.red);
        setBackground(Color.blue);
    }
    // "set"属性
    public void setOutString(String newString)
    {
        outString = newString;
    }
    // "get"属性
    public String getOutString()
    {
        return outString;
    }
}
```

2. Indexed 属性

Indexed 属性表示一个数组值，使用与该属性对应的 set/get 方法可以取得数组中的数值。该属性也可一次设置或取得整个数组的值。

Indexed 属性的用法如下：

```
public class example2 extends Canvas
{
    //dataSet 是一个 indexed 属性
    int dataSet[]={1,2,3,4,5};
    public example2()
    {
        setBackground(Color.red);
        setBackground(Color.blue);
    }
    //设置整个数组
    public void setDataSet(int x[])
    {
        dataSet = x;
    }
    //设置数组中的单个元素值
    public void setDataSet(int index,int x)
    {
        dataSet[index] = x;
    }
}
```

```
//取得整个数组值
public int getDataSet[]()
{
    return dataSet;
}
//取得数组中的指定元素值
public int getDataSet(int x)
{
    return dataSet[x];
}
}
```

代码说明：

对于 **Indexed** 属性，必须提供**两对**相匹配的 `getXXX()`与 `setXXX()`方法，一对用来设置整个数组，另一对用来获得或设定数组中的某个元素。

使用 **Indexed** 属性除了表示数组之外，还可以表示集合类。以集合类 **Map** 为例，具体代码如下：

```
import java.util.HashMap;
import java.util.Map;
public class Hello
{
    public Map map=new HashMap();
    public void setMap (Object value, Object key){    //为 Map 集合中某个键值赋值
        map.put (value, key);
    }
    public Object getMap(Object key){                //返回 Map 集合中的某个键值的数据
        return map.get (key);
    }
    public void setMap(Map map){                    //为 Map 集合赋值
        this.map=map;
    }
    public Map getMap(){                            //返回整个 Map 集合
        return map;
    }
}
```

3. Bound 属性

Bound 属性是指当该属性的值发生变化时，要通知其他的对象。每次属性值改变时，这种属性就触发一个 **PropertyChange** 事件（在 Java 程序中，事件也是一个对象）。事件中封装了属性名、属性的原值、属性变化后的新值。这种事件传递到其他的 **Bean**，至于接收事件的 **Bean** 应该做什么动作由自己定义。也就是说，**Bound** 属性提供了一种机制，即通知监听器一个 **JavaBean** 组件的属性发生了改变。监听器实现了 **PropertyChangeListener** 接口并接受由 **JavaBean** 组件产生的 **PropertyChangeEvent** 对象，**PropertyChangeEvent** 对象包括一个属性名字：旧的属性值及每个监听器可能访问的新属性值。

Bound 属性的用法如下:

```
public class example3 extends Canvas
{
    // outString 是一个 bound 属性
    String outString = "Hello";
    //声明并实例化一个 changes 对象
    private PropertyChangeSupport changes = new PropertyChangeSupport(this);
    public void setOutString(String newString)
    {
        String oldString = outString;
        outString = newString;
        //outString 的属性值已经发生变化, 于是接着激活属性改变事件
        changes.firePropertyChange("outString",oldString,newString);
    }
    public String getOutString()
    {
        return outString;
    }
    /* 以下代码是为开发工具所使用的。我们不能预知 example3 将与哪些其他的 Bean
       组合成为一个应用, 无法预知当 example3 的 outString 属性发生变化时, 有哪些其
       他的组件与此变化有关, 因而 example3 这个 Bean 要预留出一些接口给开发工具,
       开发工具使用这些接口, 把其他的 JavaBean 对象与 example3 挂接。
    */
    public void addPropertyChangeListener(PropertyChangeListener l)
    {
        changes.addPropertyChangeListener(l);
    }
    public void removePropertyChangeListener(PropertyChangeListener l)
    {
        changes.removePropertyChangeListener(l);
    }
}
```

在程序中要进行触发事件的操作, 使用的方法在 `PropertyChangeSupport` 类中。特别注意上段代码中加黑的两行: 首先声明并实例化了一个 `changes` 对象, 然后在下面使用 `changes` 的 `firePropertyChange()` 方法来触发 `outString` 的属性改变事件。

通过上面的程序代码, 开发工具调用 `changes` 的 `addPropertyChangeListener()` 方法把其他 JavaBean 注册入 `outString` 属性的监听者的队列 1 中, 队列 1 是一个 `Vector()` 数组, 可存储任何 Java 对象。开发工具也可使用 `changes` 的 `removePropertyChangeListener()` 方法, 从队列 1 中注销指定的对象, 使 `example3` 的 `outString` 属性的改变不再与这个对象有关。当然, 当程序员手写代码编制程序时, 也可以直接调用这两个方法, 把其他 Java 对象与 `example3` 挂接。

4. Constrained 属性

`Constrained` 属性是指当这个属性的值要发生变化时, 与这个属性已建立了某种连接的其他 Java 对象可否决属性值的改变。

监听器实现了 `VectorChangeListener` 接口，并接受由 `JavaBean` 组件产生的 `PropertyChangeEvent` 对象，`JavaBean` 组件可以使用 `VetoableChangeSupport` 辅助程序类激发由监听器接受的实际事件。

使用 `JavaBean` 组件实例的引用来构造 `VetoableChangeSupport` 对象，`JavaBean` 实现了用 `addVetoableChangeListener()` 方法和 `removeVetoableChangeListener()` 方法来加入或删除监听器。`VetoableChangeSupport.fireVetoableChange()` 方法可以用来传递属性的名字、旧属性值和新属性值等信息。

`Constrained` 属性有两种监听者：属性变化监听者和否决属性改变监听者。`Constrained` 属性的监听者通过抛出 `PropertyVetoException` 来阻止该属性值的改变。否决属性改变的监听者在自己的对象代码中有相应的控制语句，在监听到有 `Constrained` 属性要发生变化时，在控制语句中判断是否应否决这个属性值的改变。总之，某个 `Bean` 的 `Constrained` 属性值可否改变取决于其他的 `Bean` 或 `Java` 对象是否允许这种改变。允许与否的条件由其他的 `Bean` 或 `Java` 对象在自己的类中进行定义。

由于 `Constrained` 属性在 `JSP` 中并不多见，在此就不举例了。

6.2.2 JavaBean 的方法

`JavaBean` 处理数据的方法提供了改变 `Bean` 状态并由此采取行动的方式。如同普通的 `Java` 类一样，`Bean` 能够拥有不同访问类型的方法。例如，私有方法只有在 `Bean` 内部才可以访问，而保护方法在 `Bean` 的内部和由它派生的 `Bean` 中都可以访问。最具访问能力的方法是公共方法，它在 `Bean` 的内部从派生的 `Bean` 或者从诸如应用程序和其他组件等外界部分都可以访问。可以访问意味着应用程序能够调用组件中的任意公共方法。公共方法对于 `Bean` 来说具有独特的重要性，因为它们形成了 `Bean` 与外部环境通信的主要途径。

不论从外部看起来 `Bean` 有多么复杂，对内部它只是数据与方法的组合。

`JavaBean` 的方法是从其他组件容器或批命令环境调用的操作。`JavaBean` 的方法可以变成 `Public` 进行输出，这样就可以使用 `Java` 内置的工具浏览 `JavaBean` 的方法，用于启动或捕捉事件。

6.2.3 JavaBean 的事件

事件是 `JavaBean` 之间和 `JavaBean` 与容器之间通信的机制。`JavaBean` 通过事件进行信息的传递，事件从源听众注册或发表，并通过方法调用传递到一个或几个目标听众。事件有许多不同的用途，如在 `Windows` 系统中常要处理的鼠标事件、窗口边界改变事件、键盘事件等。在 `JavaBean` 中定义了一个一般的、可扩充的事件机制，这种机制能够实现以下功能。

- 对事件类型和传递的模型的定义和扩充提供一个公共框架，并适于广泛的应用；能完成 `JavaBean` 事件模型与相关的其他组件体系结构事件模型的中立映射。
- 事件能被扫描环境捕获和激活；能够发现指定的对象类可以观察监听到的事件。
- 能使其他构造工具采用某种技术在设计时直接控制事件，以及事件源和事件监听者之间的联系；提供一个常规的注册机制，允许动态操纵事件源与事件监听之间的关系；事件源与监听者之间可以进行高效的事件传递。

- 与 Java 语言和环境有较高的集成度；事件机制本身不依赖于复杂的开发工具；不需要其他的虚拟机和语言即可实现。

JavaBean 事件是用对象进行传递的，用户应弄清楚以下事件的内容。

- ① 事件状态对象；
- ② 事件监听者接口；
- ③ 事件监听者的注册与注销。

6.3 JavaBean 的作用域

使用<jsp:useBean>标签中的 scope 关键字可以设置 JavaBean 的 scope 属性，scope 属性决定了 JavaBean 对象的生存周期和使用范围。scope 的可选值包括 page、request、session 和 application，默认值为 page。JavaBean 的作用域和 JSP 页面的范围名称相同，意义也相同。下面分别介绍 JavaBean 的作用域。

6.3.1 作用域

1. page 作用域

当 scope 为 page 时，它的作用域在四种类型中范围最小，客户端每次请求访问时都会创建一个 JavaBean 对象。JavaBean 对象的有效范围是客户端请求访问的当前页面文件，当客户端执行完当前的页面文件后，JavaBean 对象结束生命。在 page 范围内，每次访问页面文件时都会生成新的 JavaBean 对象，原有的 JavaBean 对象已经结束生命周期。

2. request 作用域

当 scope 为 request 时，JavaBean 对象被创建后，它将存在于整个 request 的生命周期内，request 对象是一个内建对象，使用它的 getParameter 方法可以获取表单中的数据信息。request 范围的 JavaBean 与 request 对象有着很大的关系，它的存取范围除了 page 外，还包括使用动作元素<jsp:include>和<jsp:forward>包含的网页，所有通过这两个操作指令连接在一起的 JSP 程序都可以共享同一个 JavaBean 对象。

3. session 作用域

当 scope 为 session 时，JavaBean 对象被创建后，它将存在于整个 session 的生命周期内，session 对象是一个内建对象，当用户使用浏览器访问某个网页时，就创建了一个代表该链接的 session 对象，同一个 session 中的文件共享这个 JavaBean 对象。客户端对应的 session 生命周期结束时，JavaBean 对象的生命也结束了。在同一个浏览器内，JavaBean 对象就存在于一个 session 中。当重新打开新的浏览器时，就会开始一个新的 session，每个 session 中拥有各自的 JavaBean 对象。

4. application 作用域

当 scope 为 application 时，JavaBean 对象被创建后，它将存在于整个主机或虚拟主机的生命周期内，application 范围是 JavaBean 的生命周期中最长的。同一个主机或虚拟主机中的所有文件共享这个 JavaBean 对象。如果服务器不重新启动，scope 为 application 的 JavaBean 对象会一直存放在内存中，随时处理客户端的请求，直到服务器关闭，它在内存中占用的资源才会被释放。在此期间，服务器并不会创建新的 JavaBean 组件，而是创建源对象的一个同步复

制,任何复制对象发生改变都会使源对象随之改变,不过这个改变不会影响其他已经存在的复制对象。

6.3.2 获取作用域数据

为了帮助读者更好地理解作用域的含义,下面使用各种存在范围获取 `JavaBean` 中数值的例子来说明。

【例 6.1】 在各种存在范围获取 `JavaBean` 中的数值。

① 创建名称为 `Scope.java` 的 `JavaBean`,在这个类中定义了一个属性 `number` 及访问这个属性的方法:

```
package scope;
public class Scope{
    public Scope(){}           //无参的构造函数
    private int number=0;      //初始化变量 number 的值为 0
    public int getNumber(){    //增加并返回变量 number 的值
        number++;
        return number;
    }
    public void setNumber(int newNumber){
        this.number=newNumber; //给变量 number 赋新值
    }
}
```

② 创建名称为 `6_1scope.jsp` 的页面文件,该页面文件用来显示 `JavaBean` 存在的范围的具体区别,代码如下:

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title> JavaBean 存在的范围</title>
</head>
<body>
    <jsp:useBean id="pageScope" scope="page" class="scope.Scope"/>
    <% out.println("使用 page 获取的数据为: "+pageScope.getNumber());%><br>
    <jsp:useBean id="requestScope" scope="request" class="scope.Scope"/>
    <% out.println("使用 request 获取的数据为: "+requestScope.getNumber());%><br>
    <jsp:useBean id="sessionScope" scope="session" class="scope.Scope"/>
    <% out.println("使用 session 获取的数据为: "+sessionScope.getNumber());%><br>
    <jsp:useBean id="applicationScope" scope="application" class="scope.Scope"/>
    <% out.println("使用 application 获取的数据为: "+applicationScope.getNumber());%><br>
</body>
</html>
```

③ 先用 IE 浏览器运行这个程序,页面刷新 7 次后的结果如图 6.1 所示。

说明:每刷新一次页面,session 和 application 获取的数值都会同步增加 1,而 page 和 request 不变(始终保持初始值 1),读者可以试着做做看。

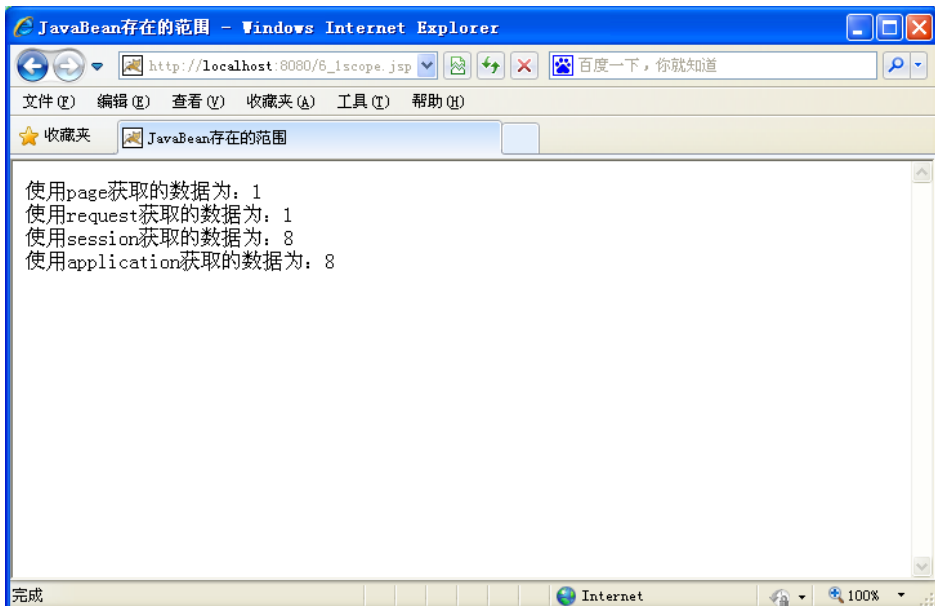


图 6.1 IE 页面刷新 7 次

④ 关闭该浏览器，用另外的浏览器（如 360 安全浏览器）运行本程序，结果如图 6.2 所示。

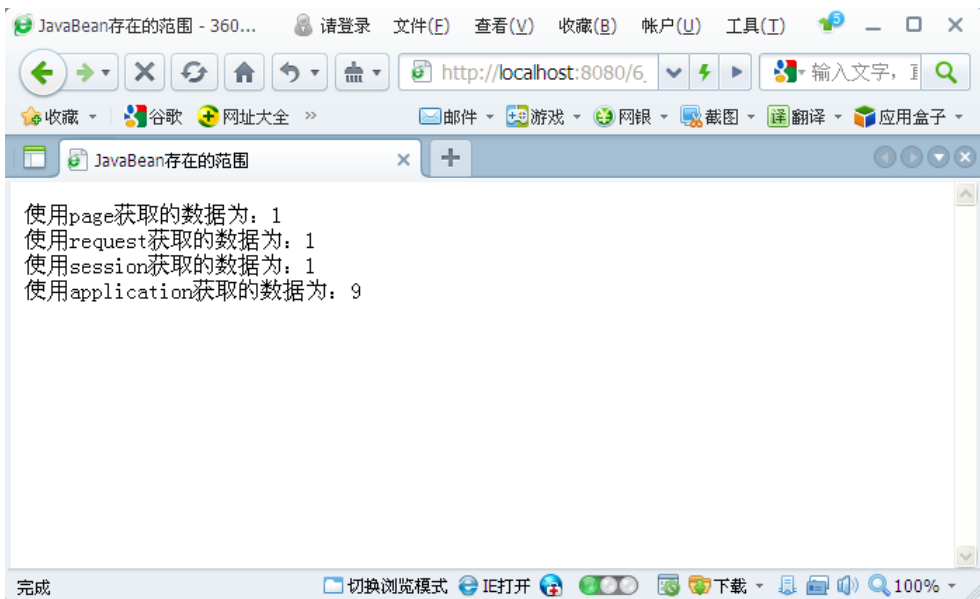


图 6.2 用 360 安全浏览器重新运行程序

大家会发现：session 的数值回归初始值 1，而 application 值仍继续增 1。这说明 session 的生命周期只维持到一个浏览器应用程序退出，而 application 的生命则是跨越浏览器应用的！若要使 application 归 1 就必须重启 Tomcat 服务器（有兴趣的读者自己试试！）。

也可以看出，page 和 request 的生命最短暂，每次刷新页面后它们都会重新归 1，在页面上看起来也就是一直维持 1 不变。

6.4 JavaBean 实现动态日历

【例 6.2】 利用 JavaBean 实现一个能进行年、月查找的日历程序。

(1) 编写 JavaBean 程序

建立一个 Bean 程序 CalendarBean.java。该 Bean 主要功能由 getCalendar()方法实现，功能包括设置当月的天数，判断大小月及闰年的情况，以数组存储每月的号码及 1 号是星期几，实现代码如下：

```
package ch6_2;
import java.util.*;
public class CalendarBean
{
    String calendar=null;
    int year=2012,month=4;
    public void setYear(int year)
    {
        this.year=year;
    }
    public int getYear()
    {
        return year;
    }
    public void setMonth(int month)
    {
        this.month=month;
    }
    public int getMonth()
    {
        return month;
    }
    public String getCalendar()
    {
        StringBuffer buffer=new StringBuffer();
        Calendar mycalendar=Calendar.getInstance();
        mycalendar.set(year,month-1,1); //将日历设置为 year 年 month 月 1 日，注意 0 表示 1 月，
                                         //以此类推，11 表示 12 月。
        //获取 1 日是星期几（get 方法返回的值是 1 表示星期日，返回的值是 7 表示星期六）
        int mydayofweek=mycalendar.get(Calendar.DAY_OF_WEEK)-1;
        int day=0;
        if(month==1||month==3||month==5||month==7||month==8||month==10||month==12)
        {
            day=31;
        }
        if(month==4||month==6||month==9||month==11)
        {
```

```

        day=30;
    }
    if(month==2)
    {
        if(((year%4==0)&&(year%100!=0))||((year%400==0)))
        {
            day=29;
        }
        else
        {
            day=28;
        }
    }
    String a[]=new String[42];        //存放号码的一维数组
    for(int i=0;i<mydayofweek;i++)
    {
        a[i]="";
    }
    for(int i=mydayofweek,n=1;i<mydayofweek+day;i++)
    {
        a[i]=String.valueOf(n);
        n++;
    }
    //用表格显示当月日历
    buffer.append("<table border=\"3\">");
    buffer.append("<tr>");
    String xingqi[]={"星期日","星期一","星期二","星期三","星期四","星期五","星期六"};
    for(int k=0;k<7;k++)
    {
        buffer.append("<td>"+xingqi[k]+"</td>");
    }
    buffer.append("</tr>");
    for(int k=0;k<42;k=k+7)
    {
        buffer.append("<tr>");        //换行
        for(int j=k;j<7+k;j++)
        {
            if(a[j]!=null)
                buffer.append("<td>"+a[j]+"</td>");
        }
        buffer.append("</tr>");
    }
    buffer.append("</table>");
    calendar=new String(buffer);
    return calendar;

```

```

    }
}

```

(2) 编写 JSP 程序

编写 6_2date.jsp 文件，在 JSP 页面中调用该 JavaBean 的方法，实现年、月的动态查找，实现代码如下：

```

<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
    <title>日历程序</title>
</head>
<body bgcolor="pink">
    <font size="4">
        <jsp:useBean id="Calendar" class="ch6_2.CalendarBean" scope="request"/>
        <form action="" method="post" name="form1">
            选择日历显示 年份:
            <Select name="year">
                <Option value="2006">2006 年
                <Option value="2007">2007 年
                <Option value="2008">2008 年
                <Option value="2009">2009 年
                <Option value="2010">2010 年
                <Option value="2011">2011 年
                <Option value="2012"selected>2012 年
            </Select>
            月份:
            <Select name="month">
                <Option value="1">1 月
                <Option value="2">2 月
                <Option value="3">3 月
                <Option value="4"selected>4 月
                <Option value="5">5 月
                <Option value="6">6 月
                <Option value="7">7 月
                <Option value="8">8 月
                <Option value="9">9 月
                <Option value="10">10 月
                <Option value="11">11 月
                <Option value="12">12 月
            </Select>
            <p>
                <input type="submit" value="提交" name="submit">
            </p>
        </form>
        <jsp:setProperty name="Calendar" property="*" />
        <font color="red"><jsp:getProperty name="Calendar" property="year"/></font>年
        <font color="red"><jsp:getProperty name="Calendar" property="month"/></font>月的日历:

```

```
<jsp:getProperty name="Calendar" property="calendar"/>
</body>
</html>
```

运行 6_2date.jsp 文件，显示结果如图 6.3 所示。



图 6.3 初始显示结果

在图 6.3 中可以看出，该日历默认日期是 2012 年 4 月，在“月份”下拉列表框中选择“10 月”，然后单击“提交”按钮，就显示 2012 年 10 月的日历，如图 6.4 所示。



图 6.4 显示 2012 年 10 月的日历

(3) 代码实现分析

① Bean 程序 CalendarBean.java。

调用了 Calendar 类的 setYear()、setMonth()、get(Calendar.DAY_OF_WEEK)等方法，其中 setYear()方法和 setMonth()方法用于设置用户输入的年、月。编写成员函数 getCalendar()实现主要功能，并输出日历。在该成员函数中定义一个 Calendar 类的对象“mycalendar”，并将日历设置为 year 年 month 月 1 日（注意 0 表示 1 月，以此类推，11 表示 12 月）。实现代码如下：

```
Calendar mycalendar=Calendar.getInstance();
mycalendar.set(year,month-1,1);
```

调用 get(Calendar.DAY_OF_WEEK)方法，返回值 1 表示星期日，返回值 7 表示星期六。定义了一个整型变量实现星期日对应整数为 0，星期六对应整数为 6。实现代码如下：

```
int mydayofweek=mycalendar.get(Calendar.DAY_OF_WEEK)-1;
```

在该 Bean 中实现用表格显示当月日历，实现代码如下：

```
buffer.append("<table border='3'>");
buffer.append("<tr>");
String xingqi[]={"星期日","星期一","星期二","星期三","星期四","星期五","星期六"};
for(int k=0;k<7;k++)
{
    buffer.append("<td>"+xingqi[k]+"</td>");
}
buffer.append("</tr>");
for(int k=0;k<42;k=k+7)
{
    buffer.append("<tr>");    //换行
    for(int j=k;j<7+k;j++)
    {
        if(a[j]!=null)
            buffer.append("<td>"+a[j]+"</td>");
    }
    buffer.append("</tr>");
}
buffer.append("</table>");
```

② JSP 程序 6_2date.jsp。

设置 userBean 语句，id 属性为“Calendar”，scope 属性为“request”，class 属性为“ch6_2.CalendarBean”。实现代码如下：

```
<jsp:useBean id="Calendar" class="ch6_2.CalendarBean" scope="request"/>
```

建立自处理表单 form，设置 method 的属性为“post”，name 的属性为“form1”，添加两个下拉列表框分别用于选择年、月，name 属性分别为“year”、“month”。

在 JSP 中使用 Bean 方法实现日历功能。实现代码如下：

```
<jsp:setProperty name="Calendar" property="*" />
<font color="red"><jsp:getProperty name="Calendar" property="year"/></font>年
<font color="red"><jsp:getProperty name="Calendar" property="month"/></font>月的日历:
<jsp:getProperty name="Calendar" property="calendar"/>
```

6.5 第三方 JavaBean 组件的应用

在 Web 开发中,程序员不仅可以自定义 JavaBean,还可以使用现成的第三方 JavaBean 组件扩充 Web 系统,增强程序的功能。Web 应用经常需要与用户进行信息交流,如上传、下载文件等操作,这些功能的实现主要依赖于 JavaBean 组件 jspSmartUpload。本节就来介绍这一组件的具体应用,演示如何将用户文件上传到服务器并保存在特定位置,以及如何从服务器端指定目录下载文件。

6.5.1 文件上传

1. jspSmartUpload 组件的下载

jspSmartUpload 组件可以直接在网上下载,压缩包的名字是 jspSmartUpload.zip。下载后,用 WinZip 或 WinRAR 将其解压缩到 Tomcat 的 webapps 目录下。解压缩后,把 webapps/jspSmartUpload 目录下的子目录名字 Web-inf 改为大写的 WEB-INF,这样一改,jspSmartUpload 类才能使用。因为 Tomcat 对文件名大小写敏感,它要求 Web 应用程序相关的类所在目录为 WEB-INF,且必须是大写。接着重新启动 Tomcat,这样就可以在 JSP 文件中使用 jspSmartUpload 组件了。

2. 编写 JSP 程序

编写 upload.htm 文件,建立表单,用来选择要上传的文件;编写 do_upload.jsp 文件,实现文件的上传。把 upload.htm 和 do_upload.jsp 保存在\webapps\jspSmartUpload 目录下。

① upload.htm 代码如下:

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
    <title>文件上传</title>
</head>
<body>
    <p align="center">上传文件选择</p>
    <form method="post" action="do_upload.jsp" enctype="multipart/form-data">
        <input type="hidden" name="TEST" value="good">
        <table width="75%" border="1" align="center">
            <tr>
                <td>
                    <div align="center">1、
                        <input type="FILE" name="FILE1" size="30">
                    </div>
                </td>
            </tr>
            <tr>
                <td>
                    <div align="center">2、
```

```

        <input type="FILE" name="FILE2" size="30">
    </div>
</td>
</tr>
<tr>
    <td>
        <div align="center">3、
            <input type="FILE" name="FILE3" size="30">
        </div>
    </td>
</tr>
<tr>
    <td>
        <div align="center">4、
            <input type="FILE" name="FILE4" size="30">
        </div>
    </td>
</tr>
<tr>
    <td>
        <div align="center">
            <input type="submit" name="Submit" value="上传它！">
        </div>
    </td>
</tr>
</table>
</form>
</body>
</html>

```

② do_upload.jsp 代码如下：

```

<%@ page contentType="text/html; charset=gb2312" language="java" %>
<%@ page import="java.util.*,com.jspsmart.upload.*" errorPage="" %>
<html>
<head>
    <title>文件上传处理页面</title>
</head>
<body>
    <%
        // 新建一个 SmartUpload 对象
        SmartUpload su = new SmartUpload();
        // 上传初始化
        su.initialize(pageContext);
        // 设定上传限制
        // 1.限制每个上传文件的最大长度
        su.setMaxFileSize(100000);
        // 2.限制总上传数据的长度
    %>

```

```

su.setTotalMaxFileSize(200000);
// 上传文件
su.upload();
// 将上传文件全部保存到指定目录
int count = su.save("/upload");
out.println(count+"个文件上传成功! ");
// 利用 Request 对象获取参数之值
out.println("TEST="+su.getRequest().getParameter("TEST")+"<br><br>");
// 逐一提取上传文件信息，同时保存文件。
for (int i=0;i<su.getFiles().getCount();i++)
{
    com.jspsmart.upload.File file = su.getFiles().getFile(i);
    // 若文件不存在则继续
    if (file.isMissing()) continue;
    // 显示当前文件信息
    out.println("<table border='1'>");
    out.println("<tr><td>表单项名 (FieldName) </td><td>"
        +file.getFieldName() + "</td></tr>");
    out.println("<tr><td>文件长度 (Size) </td><td>" +file.getSize() + "</td></tr>");
    out.println("<tr><td>文件名 (FileName) </td><td>" +file.getFileName() + "</td></tr>");
    out.println("<tr><td>文件扩展名 (FileExt) </td><td>" +file.getFileExt() + "</td></tr>");
    out.println("<tr><td>文件全名 (FilePathName) </td><td>"
        +file.getFilePathName() + "</td></tr>");

    out.println("</table><br>");
}
%>
</body>
</html>

```

3. 运行

① 先显示 upload.htm 文件，单击相对应的“浏览”按钮，选择要上传的文件，如图 6.5 所示。

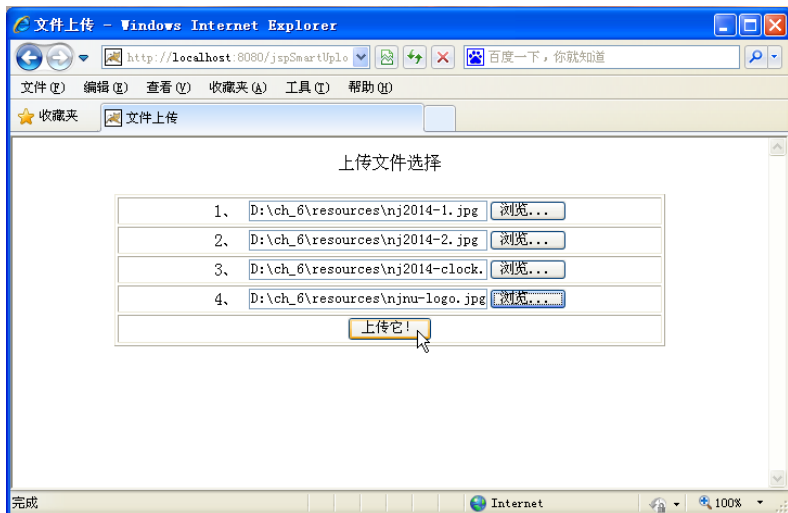


图 6.5 选择要上传的文件

② 然后单击“上传它！”按钮，显示上传成功文件的相关信息，运行结果如图 6.6 所示。



图 6.6 显示上传成功文件的相关信息

4. 代码实现分析

对于 upload.htm 文件的表单，method 属性必须为“post”，enctype 属性必须为“multipart/form-data”，否则上传不会成功。然后添加四个文件域，用于选择上传文件，即其 type 的属性为“FILE”。

对于 do_upload.jsp 文件，首先需要载入“com.jspsmart.upload.*”的 Java 组件包，然后新建一个 SmartUpload 对象 su，进行初始化。SmartUpload 还提供了一些上传限制的设定，如 setMaxFileSize()，该方法可以限制每个上传文件的最大长度（以字节为单位）；setTotalMaxFileSize()可以限制总上传数据的长度，也以字节为单位。

本例实现代码如下：

```
SmartUpload su = new SmartUpload();
su.initialize(pageContext);
su.setMaxFileSize(100000);
su.setTotalMaxFileSize(200000);
```

使用 upload()方法上传，并设置 save("/upload")，将上传文件全部保存到指定目录下。上传成功后，通过 getCount()取得上传文件数目，使用循环语句逐一显示文件信息。实现代码如下：

```

for (int i=0;i<su.getFiles().getCount();i++)
{
    com.jspsmart.upload.File file = su.getFiles().getFile(i);
    // 若文件不存在则继续
    if (file.isMissing()) continue;
    // 显示当前文件信息
    out.println("<table border=\"1\">");
    out.println("<tr><td>表单项名 (FieldName) </td><td>"
                +file.getFieldName() + "</td></tr>");
    ...
}

```

6.5.2 文件下载

上一案例上传文件后, \webapps\jspSmartUpload\upload 目录下已存在四个文件了, 如图 6.7 所示 (框出部分)。



图 6.7 已经上传的文件

编写 download.htm 文件, 建立超链接, 用来指向相应的下载文件处理程序。编写 do_download.jsp 文件, 实现 “nj2014-2.jpg” 文件的下载。

① download.htm 代码如下:

```

<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
    <title>下载文件</title>
</head>
<body>
    点击相应的链接下载<br><br>

```

```

<a href="do_download.jsp">nj2014-2.jpg</a><br><br>
</body>
</html>

```

② do_download.jsp 代码如下:

```

<%@ page contentType="text/html;charset=gb2312" %>
<%@ page import="com.jspsmart.upload.*" %>
<%
    // 新建一个 SmartUpload 对象
    SmartUpload su = new SmartUpload();
    // 初始化
    su.initialize(pageContext);
    // 设定 contentDisposition 为 null 以禁止浏览器自动打开文件
    su.setContentDisposition(null);
    // 下载文件
    su.downloadFile("/upload/nj2014-2.jpg");
%>

```

先运行 download.htm 文件, 显示结果如图 6.8 所示。

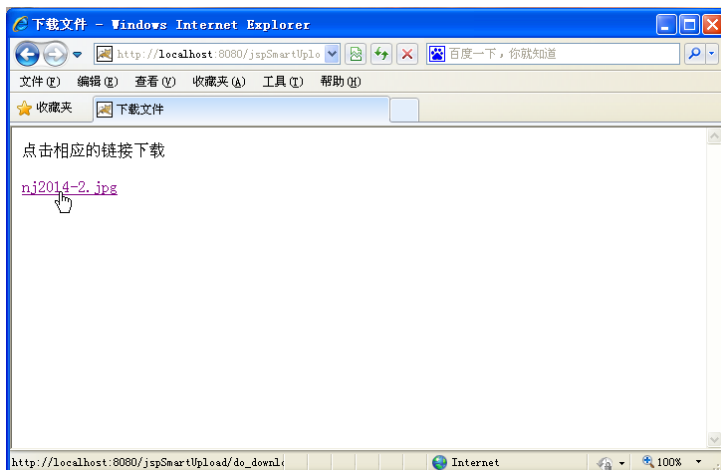


图 6.8 显示下载链接

在页面上单击“nj2014-2.jpg”超链接, 出现“文件下载”对话框, 如图 6.9 所示。

单击“保存”按钮, 就可以选择路径, 将此图片保存在指定的目录下。

对于 do_download.jsp 文件, 首先要载入所需 jspsmart 组件包, 代码如下:

```

<%@ page import="com.jspsmart.upload.*" %>

```

新建一个 SmartUpload 对象, 初始化它, 然后设定 ContentDisposition 为空, 以禁止浏览器自动打开文件, 保证单击链接后是下载文件(若不设定, 则下载的文件浏览器可能自动打开)。实现代码如下:

```

SmartUpload su = new SmartUpload();
su.initialize(pageContext);
su.setContentDisposition(null);
su.downloadFile("/upload/nj2014-2.jpg");

```

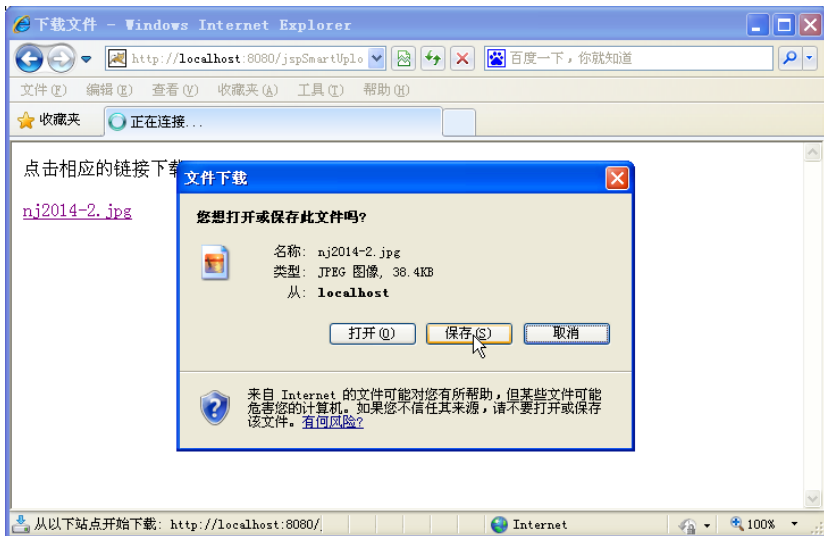


图 6.9 出现“文件下载”对话框

6.6 上机练习

1. 运行【例 6.1】的程序，观察在各种作用域内获取 JavaBean 数值的变化，页面刷新 7 次后改用 360 浏览器运行程序的结果如图 6.10 所示（与图 6.2 相同）。

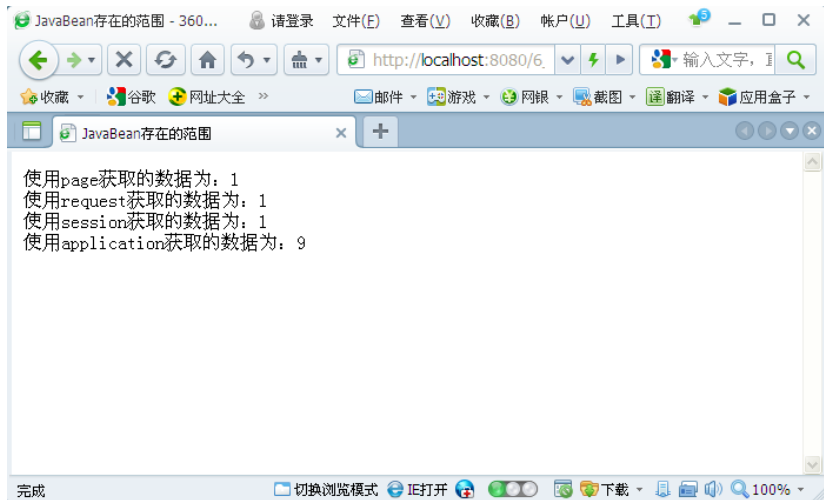


图 6.10 用 360 安全浏览器重新运行程序

请读者试着用重启 Tomcat 服务器的方法，使 application 也归 1。

2. 按照 6.5 节的指导，学会使用第三方 JavaBean 组件 jspSmartUpload 上传和下载文件。

JSP 操作数据库

大型的 Web 应用系统必须拥有强大的数据库作为后台支撑，而 JSP 程序能够非常方便地访问数据库。

7.1 数据库基础

7.1.1 关系模型

数据库按一定的数据模型组织数据，采用的模型主要有关系模型、层次模型和网状模型。**关系模型**是目前应用最广的数据模型。

关系模型以**二维表格**（关系表）的形式组织数据库中的数据。例如，学生成绩管理系统涉及学生表、课程表和成绩表（见表 7.1～表 7.3）。

学生表涉及的主要信息有学号、姓名、性别、出生日期、专业、总学分和备注。

表 7.1 学生表

学 号	姓 名	性 别	出 生 日 期	专 业	总 学 分	备 注
051101	王林	男	1991-02-10	计算机	50	
051102	程明	男	1992-02-01	计算机	50	
051103	王燕	女	1990-10-06	计算机	50	
051104	韦严平	男	1991-08-26	计算机	50	
051106	李方方	男	1991-11-20	计算机	50	
051107	李明	男	1991-05-01	计算机	54	提前修完《数据结构》，并获学分
051108	林一帆	男	1990-08-05	计算机	52	已提前修完一门课
051109	张强民	男	1989-08-11	计算机	50	
051110	张蔚	女	1992-07-22	计算机	50	三好生
051111	赵琳	女	1991-03-18	计算机	50	
051113	严红	女	1990-08-11	计算机	48	有一门课不及格，待补考
051201	王敏	男	1989-06-10	通信工程	42	
051202	王林	男	1990-01-29	通信工程	40	有一门课不及格，待补考

续表

学 号	姓 名	性 别	出 生 日 期	专 业	总 学 分	备 注
051203	王玉民	男	1991-03-26	通信工程	42	
051204	马琳琳	女	1989-02-10	通信工程	42	
051206	李计	男	1990-09-20	通信工程	42	
051210	李红庆	男	1990-05-01	通信工程	44	已提前修完一门课,并获得学分
051216	孙祥欣	男	1989-03-09	通信工程	42	
051218	孙研	男	1991-10-09	通信工程	42	
051220	吴薇华	女	1991-03-18	通信工程	42	
051221	刘燕敏	女	1990-11-12	通信工程	42	
051241	罗琳琳	女	1991-01-30	通信工程	50	转专业学习

课程表涉及的主要信息有课程号、课程名、学期、学时和学分。

表 7.2 课程表

课 程 号	课 程 名	学 期	学 时	学 分
101	计算机基础	1	64	3
102	C++程序设计	2	68	4
103	数据结构	3	68	5
104	计算机组成原理	3	96	4
105	操作系统	4	80	5
106	数据库原理	7	112	5
107	计算机网络	5	96	4
108	计算机新技术	1	32	2
201	高等数学	1	80	5
202	离散数学	3	68	4

成绩表涉及的主要信息有学号、课程号、成绩和学分。

表 7.3 成绩表

学 号	课 程 号	成 绩	学 分	学 号	课 程 号	成 绩	学 分
051101	101	80	3	051107	102	65	4
051101	102	78	4	051108	101	71	3
051102	101	66	3	051108	102	80	4
051102	102	62	4	051109	101	78	3
051103	101	70	3	051109	102	80	4
051103	102	81	4	051110	101	68	3
051104	101	90	3	051110	102	85	4
051104	102	84	4	051111	101	64	3
051106	101	65	3	051111	102	87	4
051106	102	78	4	051113	101	66	3
051107	101	78	3	051113	102	83	4

在关系表中，表格中的一行称为一个**记录**，一列称为一个**字段**，列的标题称为字段名。如果给每个关系表取一个名字，则有 n 个字段的表的结构可表示为：关系表名（字段名 1，字段名 2，...，字段名 n ），通常把关系表的结构称为**关系模式**。

在关系表中，如果一个字段或几个字段组合的值可唯一标识其对应的记录，则称该字段或字段组合为**码**。例如，表 7.1 的“学号”可唯一标识每个学生；表 7.2 的“课程号”可唯一标识每门课；表 7.3 的“学号”和“课程号”可唯一标识某个学生某门课的成绩。

有时一个表可能有多个码，比如表 7.1 中，如果姓名不重名，则“学号”、“姓名”均是学生表码。对于每个关系表，通常可指定一个码为“主码”，在关系模式中，一般用下画线标出主码。

对于表 7.1，关系模式可表示为如下形式：

学生（学号，姓名，性别，出生日期，专业，总学分，备注） //①

对于表 7.2，关系模式可表示为如下形式：

课程（课程号，课程名，学期，学时，学分） //②

对于表 7.3，关系模式可表示为如下形式：

成绩（学号，课程号，成绩，学分） //③

在数据库中，为了操作和编程方便，表名和字段名一般用代号表示。

例如，对于上面的关系模式①，用 XSB 表示学生表名，用 XH、XM、XB、CSRQ、ZY、ZXF 和 BZ 分别表示各列的字段名，写为：

XSB(XH ,XM ,XB ,CSRQ ,ZY ,ZXF ,BZ)

同样，对于关系模式②，课程表名为 KCB，课程号、课程名、学期、学时和学分对应的字段名分别为 KCH、KCM、XQ、XS 和 XF，写为：

KCB(KCH ,KCM ,XQ ,XS ,XF)

对于关系模式③，成绩表取名 CJB，学号、课程号、成绩和学分对应的字段名分别为 XH、KCH、CJ 和 XF，写为：

CJB(XH ,KCH ,CJ ,XF)

7.1.2 SQL 语言

结构化查询语言 SQL 是用于操作关系数据库的标准语言，SQL 虽名为查询语言，但实际上具有数据定义、查询、更新和控制等多种功能，它使用方便、功能丰富、简洁易学。SQL 语言由 3 部分组成。

（1）数据定义语言（DDL）

DDL 用于执行数据库定义的任务，对数据库及数据库中的各种对象进行创建、删除、修改等操作。数据库对象主要包括表、默认约束、规则、视图、触发器、存储过程。

例如，在 SQL 语言中，创建一个新数据库的基本语法格式如下：

CREATE DATABASE 数据库名称

数据库名称在服务器中必须唯一，并且符合标识符的命名规则。

此外，常用的 DDL 还有 CREATE TABLE（创建表）、CREATE PROCEDURE（新建存储过程）等。

（2）数据操纵语言（DML）

DML 用于操作数据库中的各种对象，检索和修改数据。

常用的 DML 有 INSERT（插入数据）、DELETE（删除数据）、UPDATE（更新数据）等。例如，用 INSERT 可添加记录到表中，语法如下：

```
INSERT INTO 表名 [(字段名表)] VALUES (值表)
```

现向 XSB 添加一条记录，并给所有字段赋值：

```
INSERT INTO XSB VALUES('051241', '罗林琳', '女', '1991-01-30', '通信工程', 50, '转专业学习')
```

若向 XSB 添加的记录只给其中 3 个字段赋值，则写为如下形式：

```
INSERT INTO XSB (XH, XM, ZY) VALUES('051118', '林时', '计算机')
```

至于其他 DML 及其详细的用法，本章稍后会结合实例加以介绍。

（3）数据控制语言（DCL）

DCL 用于安全管理，确定哪些用户可以查看或修改数据库中的数据。这类 SQL 语句有 GRANT、REVOKE、COMMIT、ROLLBACK 等。

7.1.3 流行的 DBMS

数据库管理系统（DataBase Management System，DBMS）是一种操纵和管理数据库的大型软件，用于建立、使用和维护数据库。目前绝大多数 DBMS 以前述的关系模型管理数据库，如 Access、Visual FoxPro、SQL Server、Sybase、Oracle、DB2 等，并且许多关系型数据库供应商都在自己的数据库中支持 SQL 语言。

下面对当下流行的几个主流的 DBMS 做简要介绍。

（1）Oracle

Oracle（中文译名“甲骨文”）是世界领先的信息管理软件开发商，因其复杂的关系数据库产品而闻名。Oracle 数据库产品为财富排行榜上的前 1 000 家公司所采用，许多大型网站也选用了 Oracle 系统。Oracle 产品引入了共享 SQL 和多线索服务器体系结构，提供了基于角色分工的安全保密管理，支持大量多媒体数据，提供了与第三代高级语言的接口软件 PRO*系列，具备了新的分布式数据库能力。

作为功能最强的重量级大型数据库产品，Oracle 主要运行于 HP-UX 和 AIX 等高端 UNIX 平台，更多地用在大型企业，随着电子商务的激增，驱使 Oracle 愈来愈成为 Web 企业级应用所需数据库的最佳选择。

目前 Oracle 最新版本为 2007 年推出的 Oracle 11g，拥有 400 多项功能！

（2）SQL Server

SQL Server 系列产品是由 Microsoft 公司开发和推广的关系数据库管理系统，是当今应用最广的关系数据库产品之一。最初由 Microsoft、Sybase 和 Ashton-Tate 三家公司共同开发。从 SQL Server 2000 开始该系列分化为企业版、标准版、个人版和开发版四种，根据不同版本的特点可以有选择地进行安装（取决于用户的业务需要），其中企业版 SQL Server 多用于大中型的产品数据库服务器，并且可以支持大型网站、企业 OLTP（联机事务处理）和大型数据仓库系统 OLAP（联机分析处理）。

Windows 平台上的数据库应用开发多选择微软 SQL Server 系列，本书 JSP 开发用的是 SQL Server 2008，稍后会有详细讲解。

（3）Visual FoxPro

Visual FoxPro 简称 VFP，是 Microsoft 公司推出的数据库开发软件，用它来开发数据库，既简单又方便。Visual FoxPro 源于美国 Fox Software 公司推出的数据库产品 FoxBase，在 DOS

上运行, 与 xBase 系列兼容。FoxPro 原来是 FoxBase 的加强版, 最高版本曾出过 2.6。之后, Fox Software 被微软收购, 加以发展, 使其可以在 Windows 上运行, 并且更名为 Visual FoxPro。目前最新版为 Visual FoxPro 9.0, 而在学校教学和教育部门考证中依然延用经典版的 Visual FoxPro 6.0。在桌面型数据库应用中, Visual FoxPro 处理速度极快, 是日常工作中的得力助手。

(4) Access

Microsoft Office Access (又名 Microsoft Access) 是由微软发布的小型数据库管理系统。它结合了 Microsoft Jet Database Engine 和图形用户界面两项特点, 是 Microsoft Office 的成员之一。Access 在 2000 年的时候成为了计算机等级考试二级选考的一种数据库语言, 并且因为它的易学易用的特点, 正逐步取代传统的 VFP 而成为二级中最受欢迎的数据库语言。

(5) DB2

DB2 是 IBM 出品的一系列关系型数据库管理系统, 能够在不同的操作系统平台上服务。虽然 DB2 产品是基于 UNIX 的系统和个人计算机操作系统, 但在基于 UNIX 系统和微软在 Windows 系统下的 Access 方面, DB2 追寻了 Oracle 的数据库产品。

(6) Informix

Informix 是 IBM 公司出品的另一个关系数据库管理系统。作为一个集成解决方案, 它被定位为 IBM 在线事务处理 (OLTP) 旗舰级数据服务系统。IBM 对 Informix 和 DB2 都有长远的规划, 两个数据库产品互相吸取对方的技术优势。在 2005 年早些时候, IBM 推出了 Informix Dynamic Server (IDS) 第 10 版。目前最新版本是 IDS11 (v11.50, 代码名为 “Cheetah 2”)。

(7) Sybase

Sybase 是美国 Sybase 公司研制的一种关系型数据库系统, 是一种典型的 UNIX 或 Windows NT 平台上客户机/服务器环境下的大型数据库系统。Sybase 提供了一套应用程序编程接口和库, 可以与非 Sybase 数据源及服务器集成, 允许在多个数据库之间复制数据, 适于创建多层应用。系统具有完备的触发器、存储过程、规则及完整性定义, 支持优化查询, 具有较好的数据安全性。Sybase 通常与 SybaseSQLAnywhere 用于客户机/服务器环境, 前者作为服务器数据库, 后者为客户机数据库。采用该公司研制的 PowerBuilder 为开发工具, 在我国大中型系统中具有广泛的应用。

(8) MySQL

MySQL 是一个小型关系型数据库管理系统, 开发商为瑞典原 MySQL AB 公司 (在 2008 年 1 月 16 日被 Sun 公司收购, 而 2009 年, Sun 又被 Oracle 收购)。MySQL 软件采用了 GPL (GNU 通用公共许可证)。由于其体积小、速度快、总体拥有成本低, 尤其是开放源码这一特点, 许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。

(9) PostgreSQL

PostgreSQL 是以加州大学伯克利分校计算机系开发的 POSTGRES (版本 4.2) 为基础的对象-关系型数据库管理系统 (ORDBMS)。PostgreSQL 支持大部分 SQL 标准并且提供了许多其他现代特性: 复杂查询、外键、触发器、视图、事务完整性、多版本并发控制。同样, PostgreSQL 可以用许多方法扩展, 例如, 通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语言。并且, 因为许可证的灵活, 任何人都可以以任何目的免费使用、修改和分发 PostgreSQL, 不管是私用、商用, 还是学术研究使用。

7.1.4 JSP 数据访问模型

与数据库的交互是动态网站的一个重要组成部分，JSP 中使用 JDBC 技术来实现与数据库的连接。

JDBC 是一种可用于执行 SQL 语句的 Java API（应用程序设计接口），它由一些 Java 语言编写的类和界面组成。JDBC 为数据库应用开发人员、数据库前台工具开发人员提供了一种标准的应用程序设计接口，使开发人员可以用纯 Java 语言编写完整的数据库应用程序。

很多数据库系统都具有 JDBC 驱动程序，JSP 可以直接利用它来访问数据库。也有一些数据库系统仅提供 ODBC 驱动程序而没有 JDBC 驱动程序，JSP 必须通过 JDBC-ODBC 桥来实现对它们的访问。

简单地说，JDBC 能实现以下三大功能。

- ① 与一个数据库建立连接。
- ② 向数据库发送 SQL 语句。
- ③ 处理数据库返回的结果。

JSP 程序借助 JDBC 访问数据库的工作原理如图 7.1 所示。

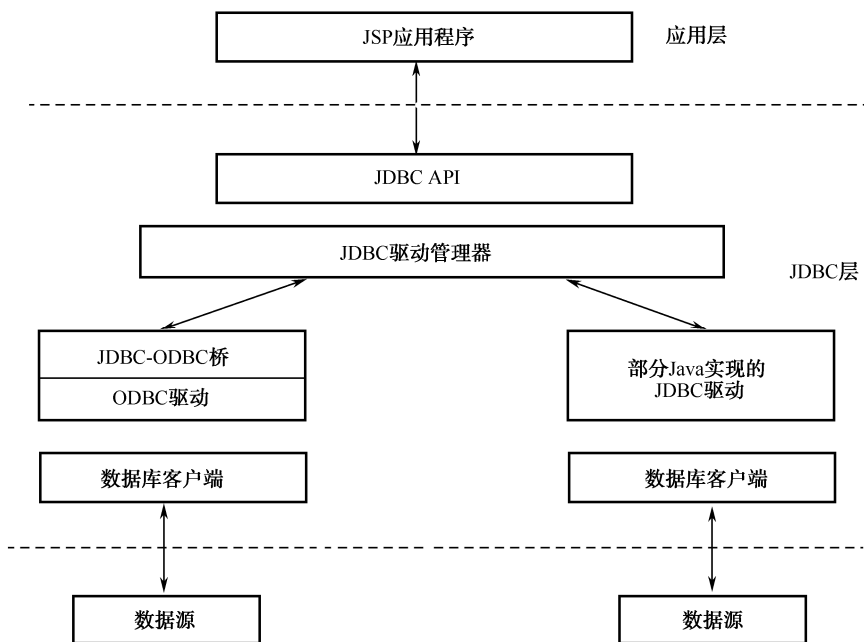


图 7.1 JSP 应用程序访问数据库的工作原理

7.2 SQL Server 2008 基础

SQL Server 2008 是 Microsoft 公司在 2008 年正式发布的一个 SQL Server 版本，是目前最新的 SQL Server 版本。作为一个重大的产品版本，它推出了许多新的特性和关键的改进，使其成为至今为止最强大、最全面的 SQL Server 数据库管理系统。

7.2.1 安装配置

SQL Server 2008 只能运行在 Windows 操作系统之上, 我们知道, 微软 Windows 自带的软件大多是以“系统服务”的组件形式存在的。SQL Server 2008 安装时需要的组件如下。

- .NET Framework 3.5。
- SQL Server Native Client。
- SQL Server 安装程序支持文件。
- Microsoft Windows Installer 4.5 或更高版本。
- Microsoft 数据访问组件 (MDAC) 2.8 SP1 或更高版本。

除此之外, 还有许多其他的 .NET 组件, 一般需要分别安装这些组件。如果机器已经安装 Visual Studio 2008, 还要进一步安装 Visual Studio 2008 SP1 补丁。这是因为 SQL Server 2008 会使用 Visual Studio 2008 SP1 补丁的某些功能 (可以理解为 .NET Framework 3.5 SP1 的功能)。

这里以 Windows XP Professional Edition SP3 操作系统为平台 (其他操作系统基本相同), 说明安装 SQL Server 2008 中文企业版时要注意的问题。

① 开始一般采用默认项。在“实例配置”窗口中进行实例配置, 如果是第一次安装, 则既可以使用默认实例, 也可以自行指定实例名称。选择“命名实例”单选按钮, 在后面的文本框中输入用户自定义的实例名称, 默认为 MSSQLSERVER, 如图 7.2 所示。

② 在实例配置完后单击“下一步”按钮进入“磁盘空间要求”窗口, 窗口中显示安装 SQL Server 2008 所需要的磁盘容量。单击“下一步”按钮进入“服务器配置”窗口, 在“服务账户”选项卡中为每个 SQL Server 服务单独配置用户名和密码及启动类型。“账户名”可以在下拉框中进行选择, 也可以单击“对所有 SQL Server 服务器使用相同账户”按钮, 为所有的服务分配一个相同的登录账户。配置完成后的界面如图 7.3 所示, 单击“下一步”按钮。

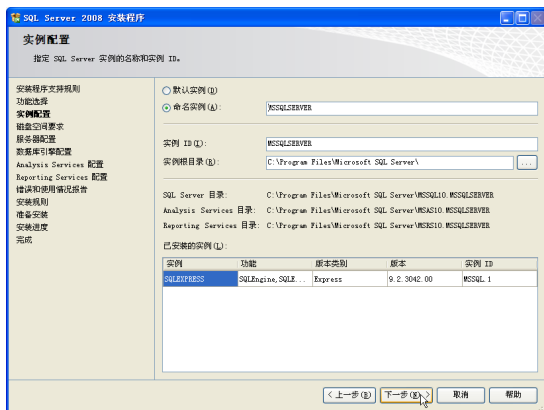


图 7.2 实例配置窗口

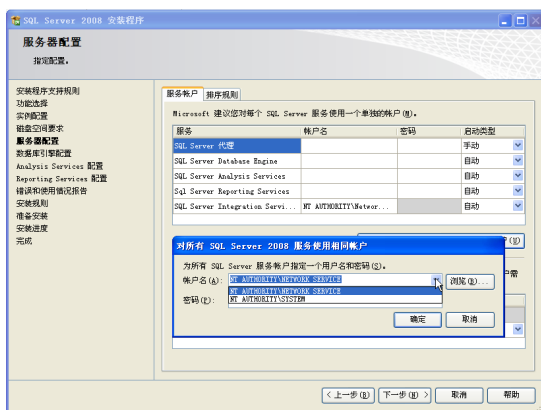


图 7.3 服务器配置窗口

③ 进入“数据库引擎配置”窗口, 在“账户设置”选项卡中选择身份验证模式。身份验证模式是一种安全模式, 用于验证客户端与服务器的连接, 它有两个选项: Windows 身份验证模式和混合模式。在 Windows 身份验证模式中, 用户通过 Windows 账户连接时, 使用 Windows 操作系统中的信息验证账户名和密码, 混合模式允许用户使用 Windows 身份验证或 SQL Server 身份验证进行连接, 而建立连接后, 系统的安全机制对于两种连接是一样的。

本书选择“混合模式”为身份验证模式，并为内置的系统管理员账户“sa”设置密码，为了便于介绍，这里将密码设为“123456”，如图 7.4 所示。在实际操作过程中，密码要尽量复杂以提高安全性。

④ 必须为 SQL Server 实例指定至少一个 SQL Server 管理员，单击“添加当前用户”按钮添加当前 Windows 账户为 SQL Server 管理员，若要添加其他账户可以单击“添加”按钮。在“数据库引擎配置”窗口的“数据目录”选项卡中还可以修改各种数据库的安装目录和备份目录，这里使用默认的目录。

单击“下一步”按钮进入“Analysis Services 配置”窗口，对 Analysis Services 进行设置，如图 7.5 所示，单击“添加当前用户”按钮指定当前 Windows 登录用户对 Analysis Services 具有管理权限。单击“下一步”按钮进入“Reporting Services 配置”窗口，选择“安装本机模式默认配置”选项，单击“下一步”按钮进入“错误和使用情况报告”窗口，这里用户可以根据需求在复选框中选择选项。

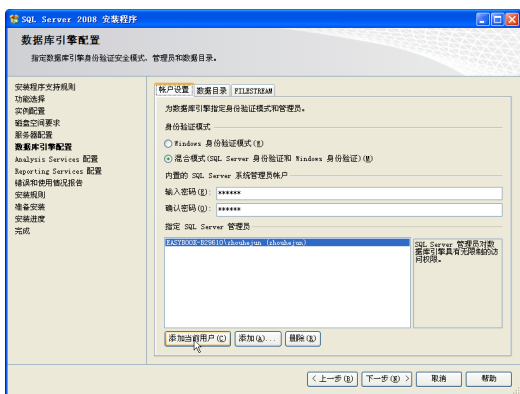


图 7.4 身份验证模式选择

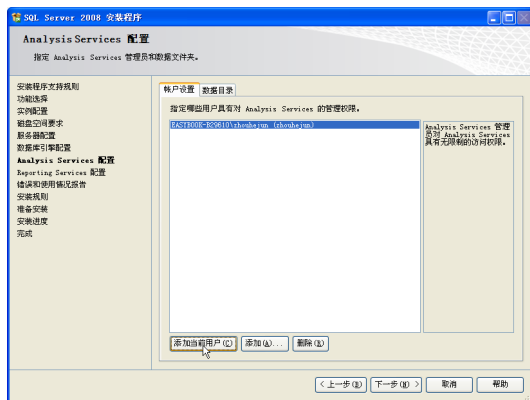


图 7.5 Analysis Services 配置

其他采用默认项，直到安装完成，窗口中将显示已经成功安装的功能组件，如图 7.6 所示。单击“下一步”按钮，在“完成”窗口中单击“关闭”按钮结束安装。

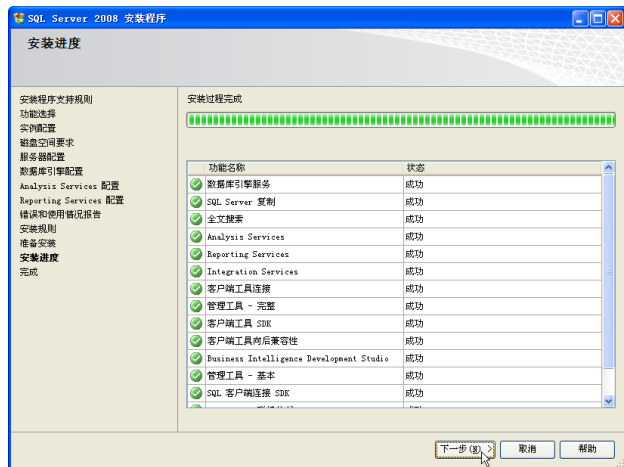


图 7.6 已成功安装的功能组件

7.2.2 SQL Server 2008 服务器组件

SQL Server 2008 是一个功能全面整合的数据平台,它包含了数据库引擎(Database Engine)、Analysis Services、Integration Services 和 Reporting Services 等组件。SQL Server 2008 的版本不同,提供的组件可能也不相同。

SQL Server 2008 服务器组件可由 SQL Server 配置管理器启动、停止或暂停。这些组件在 Windows 操作系统上作为服务运行。

(1) Database Engine

数据库引擎是 SQL Server 2008 用于存储、处理和保护数据的核心服务。例如,查询数据、创建数据库、创建表和视图等操作都是由数据库引擎完成的。数据库引擎还提供了受控访问和快速事务处理功能,并提供了大量支持以保持可用性。Service Broker(服务代理)、Replication(复制技术)和 Full Text Search(全文搜索)都是数据库引擎的一部分。

SQL Server 2008 支持在同一台计算机上同时运行多个 SQL Server 数据库引擎实例。每个 SQL Server 数据库引擎实例各有一套不为其他实例共享的系统及用户数据库,应用程序连接同一台计算机上的 SQL Server 数据库引擎实例的方式,与连接其他计算机上运行的 SQL Server 数据库引擎的方式基本相同。SQL Server 2008 实例有两种类型。

① 默认实例。SQL Server 2008 默认实例仅由运行该实例的计算机的名称唯一标识,它没有单独的实例名,默认实例的服务名称为 MSSQLSERVER。如果应用程序在请求连接 SQL Server 时只指定了计算机名,则 SQL Server 客户端组件将尝试连接这台计算机上的数据库引擎默认实例。一台计算机上只能有一个默认实例,而默认实例可以是 SQL Server 的任何版本。

② 命名实例。除默认实例外,所有数据库引擎实例都可以由在安装该实例的过程中指定的实例名标识。应用程序必须提供准备连接的计算机的名称和命名实例的实例名。计算机名和实例名格式为“计算机名\实例名”,命名实例的服务名称即为指定的实例名。

(2) Analysis Services

分析服务(SQL Server Analysis Services, SSAS)为商业智能应用程序提供联机分析处理(OLAP)和数据挖掘功能。

(3) Integration Services

集成服务(SQL Server Integration Services, SSIS)主要用于清理、聚合、合并、复制数据的转换,以及管理 SSIS 包。除此之外,它还提供生产并调试 SSIS 包的图形向导工具,以及用于执行 FTP 操作、电子邮件消息传递等工作流功能的任务。

(4) Reporting Services

报表服务(SQL Server Reporting Services, SSRS)是基于服务器的报表平台,可以用来创建和管理包含关系数据源和多维数据源中的数据的表格、矩阵、图形和自由格式的报表。该服务的进程端口与 Tomcat 有冲突,经常影响 Tomcat 的正常工作,故运行 JSP 程序前要将其关停。

7.2.3 Management Studio 环境

SQL Server 2008 使用的图形界面管理工具是“SQL Server Management Studio”(简称为 SSMS)。这是一个集成的统一的管理工具组,在 SQL Server 2005 版本之后已经开始使用这个

工具组开发、配置 SQL Server 数据库，发现并解决其中的故障。SQL Server 2008 继续使用这个工具组，并对其进行了一些改进。

在“SQL Server Management Studio”中主要有两个工具：对象资源管理器（图形化的管理工具）和查询分析器（Transact SQL 编辑器）。此外还拥有“解决方案资源管理器”窗口、“模板资源管理器”窗口和“注册服务器”窗口等。在此仅介绍本书要使用的**对象资源管理器**。

打开“SQL Server Management Studio”（即启动 SQL Server 2008）的方法如下。

在桌面上单击“开始”→“所有程序”→“Microsoft SQL Server 2008”→“SQL Server Management Studio”菜单项，在出现的“连接到服务器”对话框中输入之前安装时设置的密码（123456），单击“连接”按钮，如图 7.7 所示，就可以以 SQL Server 身份验证模式启动 SQL Server Management Studio，并以计算机系统管理员身份连接到 SQL Server 服务器。



图 7.7 启动 SQL Server 2008

启动后，用户可以直接通过 SQL Server 2008 的“对象资源管理器”窗口来操作数据库，如图 7.8 所示。

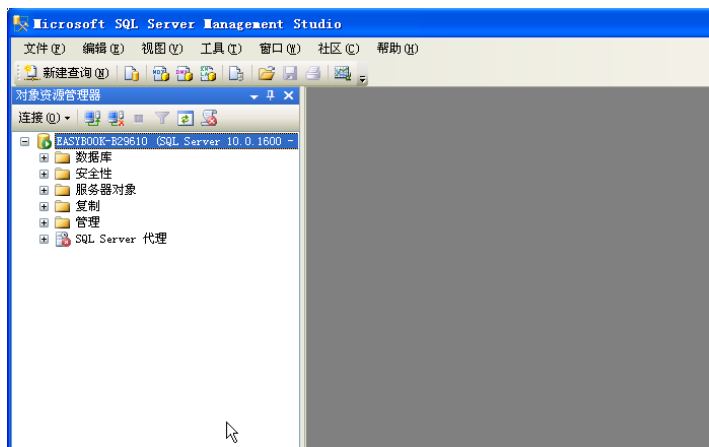


图 7.8 SQL Server 2008 的“对象资源管理器”窗口

7.2.4 建立数据库和表

观察 SQL Server Management Studio 中的“对象资源管理器”窗口可以发现，在对象资源管理器中可以浏览所有的数据库及其对象。

在对象资源管理器中展开“数据库”，例如，选择“系统数据库”中的 master 数据库并展开，则将列出该数据库中包含的所有对象，如表、视图、存储过程等，表中的各个字段及其类型都看得一清二楚，如图 7.9 所示。

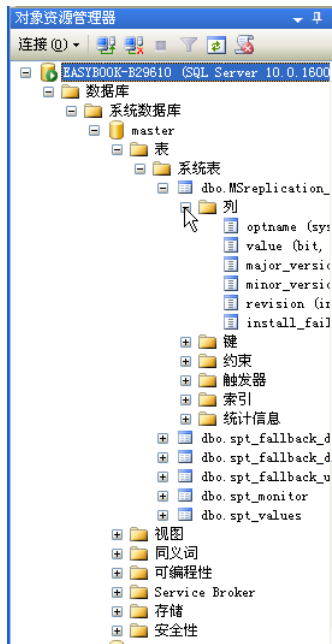


图 7.9 浏览数据库对象

1. 建立数据库

在对象资源管理器中右击“数据库”，选择“新建数据库”菜单项，如图 7.10 所示。

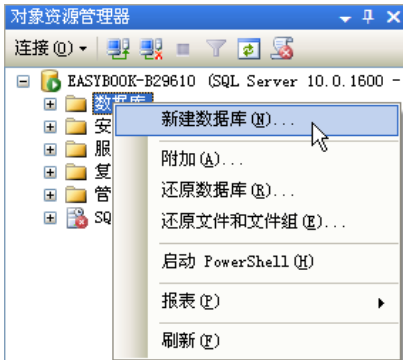


图 7.10 新建数据库

弹出如图 7.11 所示的“新建数据库”窗口，输入数据库名称 XSCJ（本书所用的数据库），单击“确定”按钮完成数据库的创建。

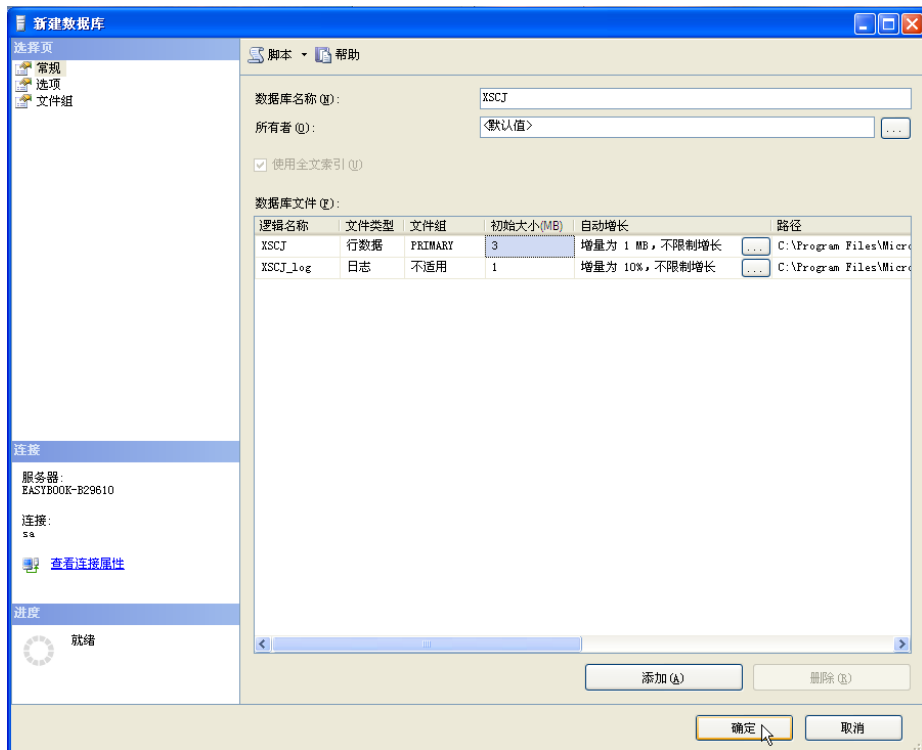


图 7.11 创建 XSCJ 数据库

完成后可以看到对象资源管理器的“数据库”目录树下多了 XSCJ 一项，如图 7.12 所示（框出部分）。

2. 建立表

展开 XSCJ 数据库，右击“表”，选择“新建表”菜单项，如图 7.13 所示。

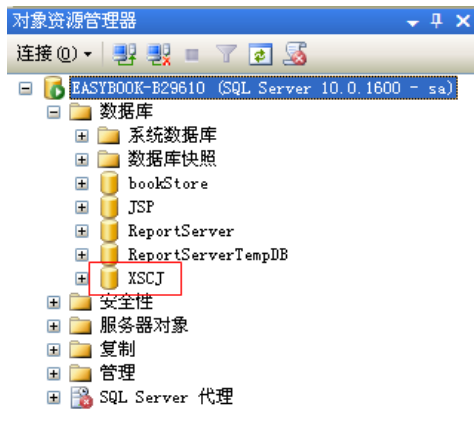


图 7.12 创建成功

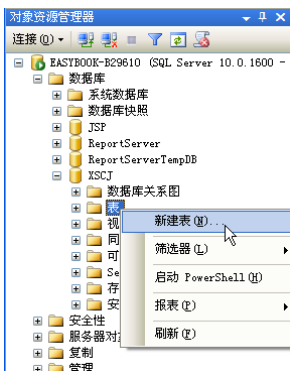


图 7.13 新建表

此时主界面中央出现了表编辑区（图 7.14 中框出部分），用户可以编辑表的各个字段。

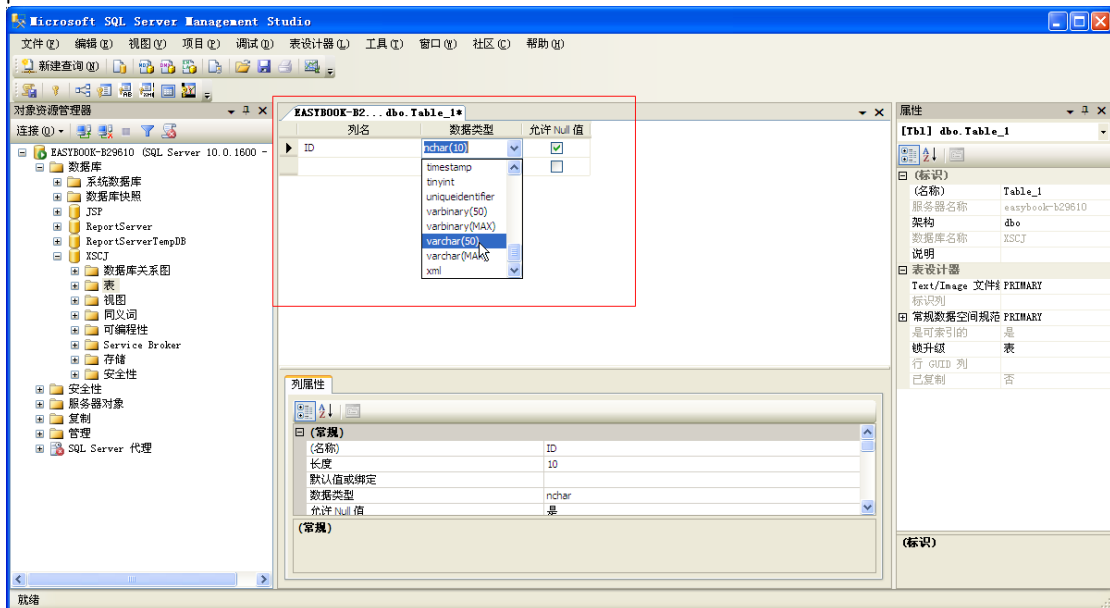


图 7.14 编辑数据库表的各个字段

请读者参照 7.1.1 节表 7.1~7.3 的内容，创建本章实例将要用到的各个表（当然也可以在以后做实例用到时再创建），并录入数据。

7.3 JDBC 连接 SQL Server 数据库

JSP 程序需要通过 JDBC 接口才能访问数据库，如图 7.1 所示，而 SQL Server 2008 本身就具有配套的 JDBC 驱动程序，JSP 可以直接利用它。上网下载 SQL Server 2008 的数据库驱动 `sqljdbc4.jar`，保存在一个特定目录下，笔者存盘在 `D:\TDDOWNLOAD` 下。在使用这个驱动之前，要先建立数据源连接。

7.3.1 在 MyEclipse 中创建连接

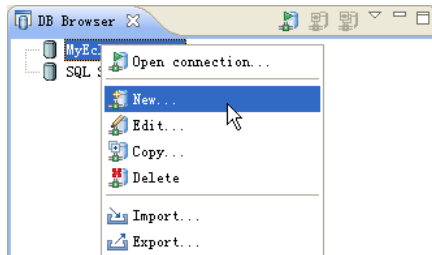


图 7.15 创建一个新的连接

在 MyEclipse 中创建对 SQL Server 2008 的数据源连接十分方便。

启动 MyEclipse，选择“Window”→“Open Perspective”→“MyEclipse Database Explorer”菜单项，打开 MyEclipse Database 浏览器，右击鼠标，如图 7.15 所示，选择“New...”菜单项，出现如图 7.16 所示的窗口，编辑数据库连接驱动。

在“Driver name”栏中填写要建立连接的名称，命名为“SQL SERVER 2008”；在“Connection URL”栏中输入要连接数据库的 URL 为“`jdbc:sqlserver://localhost:1433;databaseName=XSCJ`”；在“User name”栏中输入连接数据库的用户名；在“Password”栏中输入连接数据库的密码（笔者机器

上数据库的登录名为 sa，密码为 123456，都是在之前安装 SQL Server 2008 时设置的)。在“Driver JARs”栏中单击“Add JARs”按钮，找到数据库驱动（即先前的存盘路径 D:\TDDOWNLOAD）。

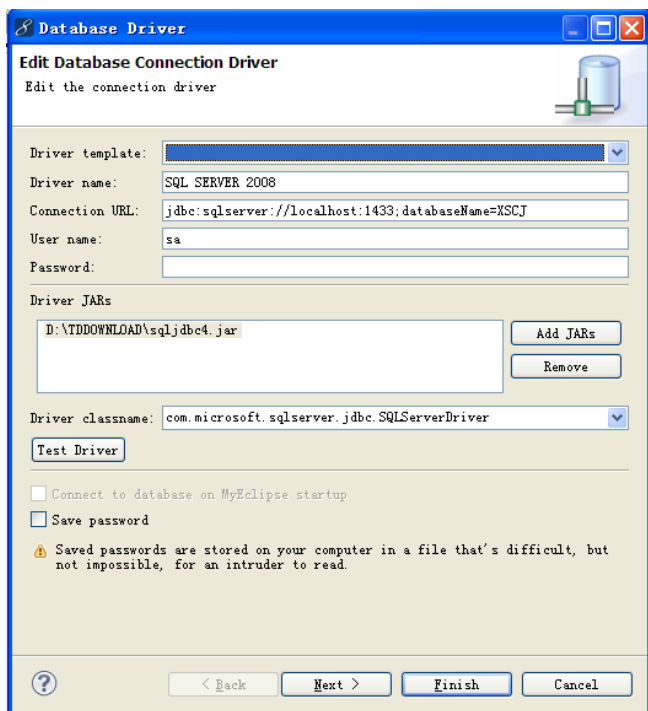


图 7.16 编辑数据库连接驱动

编辑完成以后，在 MyEclipse Database 浏览器中，右击刚才创建的 SQL SERVER 2008 数据库连接，选择“Open connection...”菜单项，打开这个数据库连接，如图 7.17 所示。

打开连接时需要输入密码，如图 7.18 所示，输入“123456”后单击“OK”按钮。

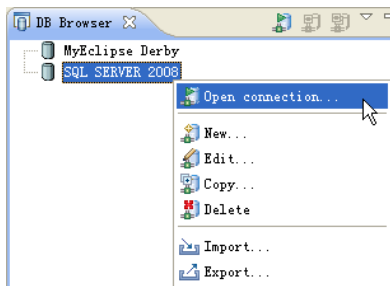


图 7.17 打开数据库连接

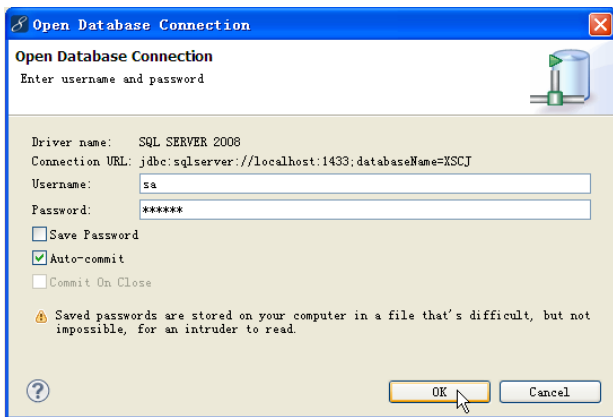


图 7.18 输入连接密码

连接打开之后，可以看到数据库中的表，就像在 SQL Server Management Studio 对象资源管理器中看到的一样，如图 7.19 所示，这就说明 MyEclipse 已经成功与 SQL Server 2008 相连了！

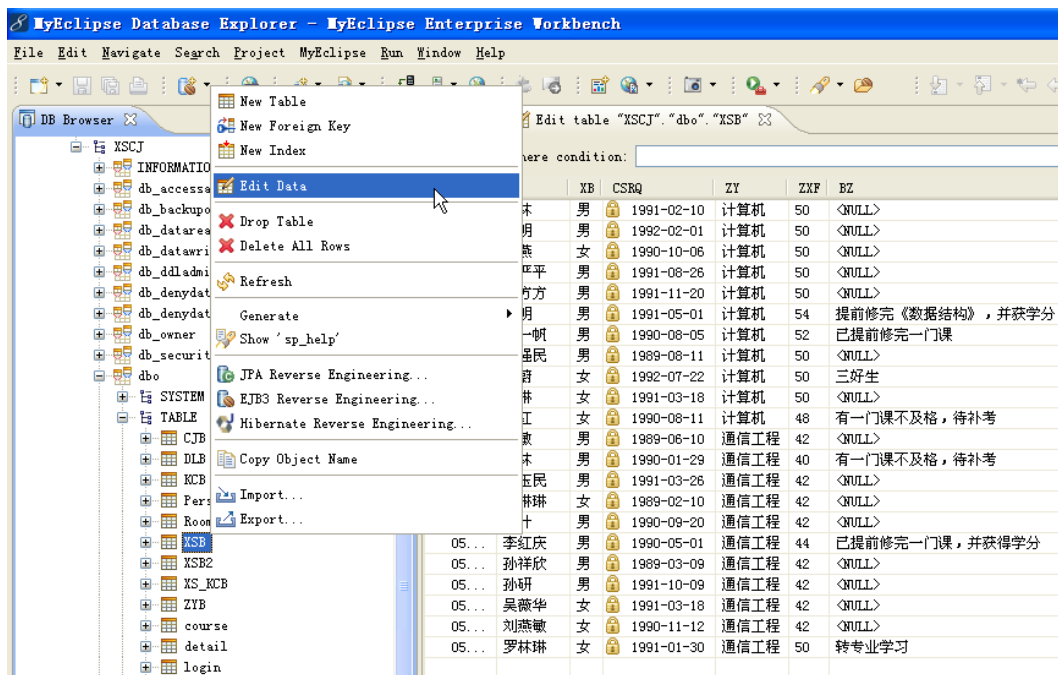


图 7.19 从 MyEclipse Database 浏览器访问数据库

今后大家在做例子的时候，可以直接使用这个现成的连接。

7.3.2 解决 Tomcat 与 SQL Server 2008 端口冲突

在安装完 SQL Server 2008 后，做 JSP 开发时经常会遇到 Tomcat 无法正常启动的问题，这是由于它们二者的端口冲突导致的，解决办法如下。

在 Windows 命令行模式下输入 netstat -ano，如图 7.20 所示。

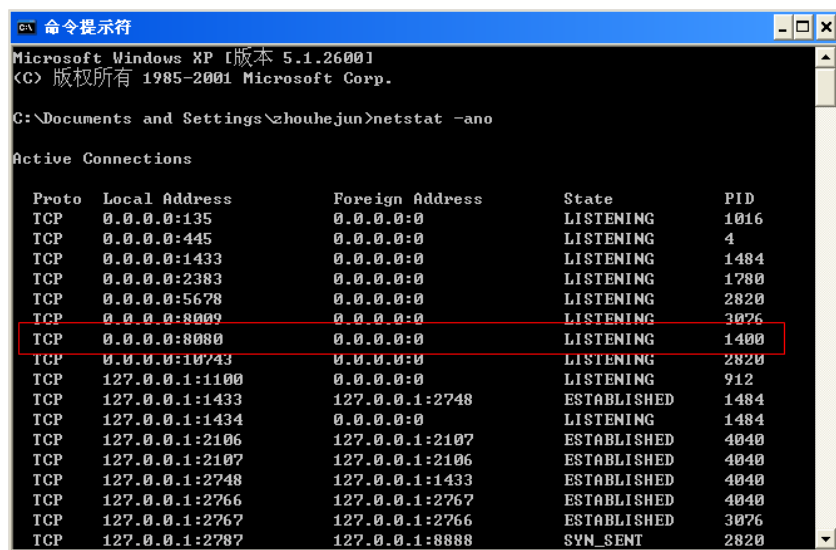


图 7.20 查找冲突进程

系统会列出当前正在运行的全部进程的信息，其中“Local Address”一项为进程占用的端口，由于 Tomcat 默认使用 8080 端口，图中如果有进程也使用这个端口的即为 SQL Server 2008 的冲突进程（见图 7.20 中框出部分），得知其进程 PID 为 1400（这个号会经常变化，不是固定值）。

在桌面任务栏上右击鼠标，打开“Windows 任务管理器”窗口，如图 7.21 所示，找到 PID 为 1400 的进程，原来是 Reporting Services 服务，它是 7.2.2 节提到的 SQL Server 2008 服务器组件之一，这个组件经常会在用户开机时自动启动，妨碍 Tomcat 的正常运行，给 JSP 开发初学者带来极大的困扰，而它本身又对 JSP 开发毫无用处，所以将其选中后单击“结束进程”按钮关掉即可。



图 7.21 关掉无用的 Reporting Services 服务

这样 Tomcat 就能正常运行了。今后读者一旦遇到 Tomcat 无法正常启动的情况，应该首先检查是不是这个程序捣的鬼。

7.3.3 测试连接的可用性

下面用编程的方法来测试这个数据库连接的可用性。

启动 MyEclipse，创建 Web 项目，将工程命名为“JdbcToSqlServer”。

1. 创建操作数据库的 JavaBean

在 src 目录下创建包 xscj_bean，在包中创建类 SQLServerConnBean，该类实现了一个操作数据库的 JavaBean。

代码如下：

```
package xscj_bean;
import java.sql.*;
public class SQLServerConnBean {
    private Statement stmt = null;
    private Connection conn = null;
```



```
ResultSet rs = null;
//构造函数
public SQLServerConnBean () { }
public void OpenConn()throws Exception
{
    try
    {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
        String url = "jdbc:sqlserver://localhost:1433;databaseName=XSCJ";
        String user = "sa";
        String password = "123456";
        conn = DriverManager.getConnection(url,user,password);
    }
    catch(SQLException e)
    {
        System.err.println("Data.executeQuery: " + e.getMessage());
    }
}
//执行查询类的 SQL 语句，有返回集
public ResultSet executeQuery(String sql)
{
    rs = null;
    try
    {
        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE
                                     ,ResultSet.CONCUR_UPDATABLE);
        rs = stmt.executeQuery(sql);
    }
    catch(SQLException e)
    {
        System.err.println("Data.executeQuery: " + e.getMessage());
    }
    return rs;
}
//关闭对象
public void closeStmt()
{
    try
    {
        stmt.close();
    }
    catch(SQLException e)
    {
        System.err.println("Date.executeQuery: " + e.getMessage());
    }
}
```

```

public void closeConn()
{
    try
    {
        conn.close();
    }
    catch(SQLException e)
    {
        System.err.println("Data.executeQuery: " + e.getMessage());
    }
}
}

```

2. 编写 JSP 程序

将 SQLServerConnBean 创建好之后,编写 SQLServer.jsp 文件,该文件是通过 SQLServerConnBean 访问 XSCJ 数据库中的表 XSB 的 JSP 程序。

```

<%@ page contentType="text/html;charset=gb2312"%>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="SqlBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=gb2312">
    <title>JDBC 连接 SQL Server 数据库</title>
    <style type="text/css">
    <!--
        .style1 {
            color:#ff0000;
            font-size:24px;
        }
    -->
    </style>
</head>
<body>
    <div align="center">
        <span class="style1">JDBC 连接 SQL Server 数据库</span><br><hr><br>
    </div>
    <table border="2" bordercolor="#ffcccc" align="center">
        <tr bgcolor="cccccc" align="center">
            <td>学号</td>
            <td>姓名</td>
            <td>专业</td>
            <td>总学分</td>
        </tr>
        <%
            //查询 XSB 表中 XH,XM,ZY 和 ZXF 字段的前 10 条记录
            String sql="select top 10* from XSB";
            SqlBean.OpenConn();
            //调用 SqlBean 中加载 JDBC 驱动的成员函数

```

```

        ResultSet rs=SqlBean.executeQuery(sql);    //取得结果集
        while(rs.next())
        {
    %>
    <tr>
        <td><%=rs.getString("XH")%></td>
        <td><%=rs.getString("XM")%></td>
        <td><%=rs.getString("ZY")%></td>
        <td><%=rs.getInt("ZXF")%></td>
    </tr>
    <%
        }
    %>
    <%
        out.print("数据库操作成功，恭喜你！");
        rs.close();
        SqlBean.closeStmt();
        SqlBean.closeConn();
    %>
    </table>
</body>
</html>

```

运行结果如图 7.22 所示，能够顺利地访问到 XSCJ 数据库中的学生信息，说明连接是可用的。

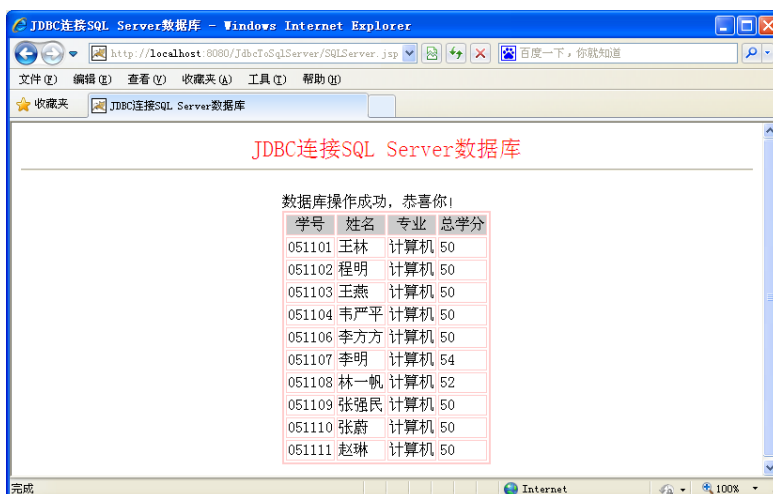


图 7.22 数据库操作成功

3. 程序说明

上面的程序利用 JavaBean 通过 JDBC 访问 SQL Server 数据库，下面对其代码进行详细说明。

(1) SQLServerConnBean.java 代码

- java.sql 包。java.sql 包中包含了所有与数据库相关的对象和方法，并且包含了使用 Java 操作数据库的类和接口，所以在使用 JDBC 的程序中必须使用“import java.sql.*”语句导入 java.sql 包。

- 加载驱动程序。在 JDBC 中，通常有两种加载驱动程序的方式。一种是将驱动程序加载到 `java.lang.System` 的属性 `jdbc.drivers` 中。这是一个由 `DriverManager` 类加载的驱动程序类名的列表，用冒号分隔。在 JDBC 中的 `java.sql.DriverManager` 类初始化时，在 JVM 的系统属性中搜索 `jdbc.drivers` 字段的内容。如果存在以冒号分隔的驱动程序名称，则 `DriverManager` 类加载相应的驱动程序。另一种方式是在程序中利用 `Class.forName()` 方法加载指定的驱动程序，这样将显式地加载驱动程序。第二种方式与外部设置无关，因此推荐使用这种加载驱动程序的方式。实现代码如下：

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
```

其中 `com.microsoft.sqlserver.jdbc.SQLServerDriver` 为 SQL Server 的 JDBC 驱动程序的别名。

- 创建指定数据库的 URL。要建立与数据库的连接，首先要创建指定数据库的 URL。数据库 URL 对象与网络资源的统一资源定位类似，具体语法如下：

```
jdbc:subProtocol:subName://hostname:port;databaseName=dbname
```

其中 `jdbc` 表示当前通过 Java 的数据库连接进行数据库的访问；`subProtocol` 表示通过某种驱动程序支持的数据库连接机制；`subName` 表示在当前连接机制下所采用驱动的具体名称；`hostname` 表示主机名；`port` 表示相应的连接端口；`databaseName` 是要连接的数据库的名称。

上例创建数据库 XSCJ 的 URL 的代码如下：

```
String url = "jdbc:sqlserver://localhost:1433;databaseName=XSCJ";
```

它利用 Microsoft 提供的机制，选择名称为 `sqlserver` 的驱动，通过 1433 端口访问本机上的 XSCJ 数据库。

- 建立与数据库的连接。连接通常是通过数据库的 URL 对象，利用 `DriverManager` 的 `getConnection()` 方法建立的。创建数据库连接时，需要提供数据库的 URL 和驱动类型，并且提供访问数据库的用户名和密码。如果有多个 JDBC 驱动程序可以与给定的 URL 连接，`DriverManager` 将轮流在每个驱动程序上调用方法 `Driver.connect`，并将用户传递给方法 `DriverManager.getConnection()` 的 URL 传递给它们，对这些驱动程序进行测试，然后连接第一个可以成功连接到给定 URL 的驱动程序。

上例程序中实现数据库连接的代码如下：

```
String user = "sa";  
String password = "123456";  
conn = DriverManager.getConnection(url,user,password);
```

其中，`DriverManager.getConnection()` 方法中的三个参数 `url`, `user`, `password` 分别指定了要连接的数据库的 URL 地址、登录的用户名和密码。

- 访问数据库。连接到数据库后就可以访问数据库了。首先要使用 `Connection` 类对象的 `createStatement` 方法从指定的数据库连接，得到一个 `Statement` (`java.lang` 包) 的实例 “`stmt`”，然后使用这个实例的 `executeQuery()` 方法来执行 SQL 语句。实现代码如下：

```
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
```

`java.sql.ResultSet` 类用来保存 SQL 语句的执行结果，还可以存取结果中的数据。通常对数据库查询或修改等操作将返回一个包含查询结果的 `ResultSet` 对象。这里将从 XSB 中查询的结果保存在 `ResultSet` 对象 `rs` 中，代码如下：

```
rs = stmt.executeQuery(sql);
```

字符串变量“sql”是 JSP 页面要执行的 SQL 语句，通过“sql”变量得到相应的 ResultSet 对象。

- 关闭数据库连接，释放资源。对数据库访问结束后，及时地关闭 ResultSet 对象、Statement 对象和 Connection 对象，从而释放它们所占的资源。释放这些资源需要使用 close() 方法，实现代码如下：

```
rs.close();      //关闭 ResultSet 对象
stmt.close();    //关闭 Statement 对象
conn.close();    //关闭 Connection 对象
```

(2) JSP 页面的代码

获取数据库中的记录并输出。数据库查询的结果保存在 ResultSet 对象中，所以在 JSP 页面中定义一个 ResultSet 对象“rs”，通过访问 SqlBean 的 executeQuery() 方法，取得数据库检索后的结果集。具体代码如下：

```
ResultSet rs=SqlBean.executeQuery(sql);
```

ResultSet 对象可以包含任意数量的列，可以直接按名称访问这些列；同时它又包含一行或多行记录，可以按照顺序从前往后或从后往前逐一访问这些行。上例中获取数据库中的记录并输出的代码如下：

```
<%
    while(rs.next())
    {
%>
        <tr>
            <td><%=rs.getString("XH")%></td>
            <td><%=rs.getString("XM")%></td>
            <td><%=rs.getString("ZY")%></td>
            <td><%=rs.getInt("ZXF")%></td>
        </tr>
    <%
    }
%>
```

其中 ResultSet 的 next() 方法用来使记录指针指向数据结果中的下一条记录，next() 方法返回一个布尔型的值，标识是否能够定位到下一条记录，如果返回值为 false，则表示不存在可以提取的记录。ResultSet 的 getString() 方法用来获取当前记录的某个字段的值，返回类型为 String 类型；getInt() 方法用来返回一个 Int 类型的值。

从本例可以看出，JavaBean 只负责执行数据库操作，不关心显示方面的逻辑，这样就有效地实现了显示逻辑和业务逻辑的分离，使程序的结构更加清晰。

7.4 JSP 操作 SQL Server 数据库

借助 JDBC，我们已能成功地连接到数据库，接下来就以上一节已经建好了的项目工程 JdbcToSqlServer 为基础，编写一些 JSP 程序实例，来完成数据库记录的增、删、改、查等常规操作。

7.4.1 添加记录

【例 7.1】在 JSP 页面中利用 INSERT 语句向 KCB（课程表）添加一门新课程“微机原理”。

（1）添加成员方法

因为本例涉及向数据库表中写入数据，必须在操作数据库的 JavaBean 类 SQLServerConnBean 中增加一个成员方法 executeUpdate()。

增加的成员方法的代码如下：

```
public void executeUpdate(String sql)
{
    stmt = null;
    try
    {
        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE
                                     ,ResultSet.CONCUR_UPDATABLE);

        stmt.executeUpdate(sql);
        stmt.close();
        conn.close();
    }
    catch(SQLException e)
    {
        System.err.println("Data.executeUpdate: " + e.getMessage());
    }
}
```

这个新增的成员方法不仅可以用于执行 INSERT 语句，还适用于 UPDATE、DELETE 语句及其他 SQL DDL，如 CREATE TABLE 和 DROP TABLE。该方法的返回值是一个整数，指示更新行数。对于 CREATE TABLE 或 DROP TABLE 等不操作行的语句，方法的返回值总为零。

（2）向数据库中添加新的记录

这里通过 SQLServerConnBean 的 executeUpdate()成员方法，编写 JSP 程序 7_1addRecord.jsp：

```
<%@ page contentType="text/html; charset=gb2312" %>
<%@ page import="java.sql.*"%>
<jsp:useBean id="AddRecordBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<head>
    <title>课程表中添加记录</title>
</head>
<body>
    <center>
        <font size="5" color="blue">课程表中添加记录</font>
    </center><br><hr><br>
    <%
        String sql="INSERT INTO KCB VALUES ('109','微机原理',1,80,5);"
```

```

        AddRecordBean.OpenConn();           //加载驱动，连接数据库
        AddRecordBean.executeUpdate(sql);    //更新数据库
        out.println("恭喜，记录插入成功！");

    %>
</body>
</html>

```

(3) 运行程序

运行结果如图 7.23 所示。

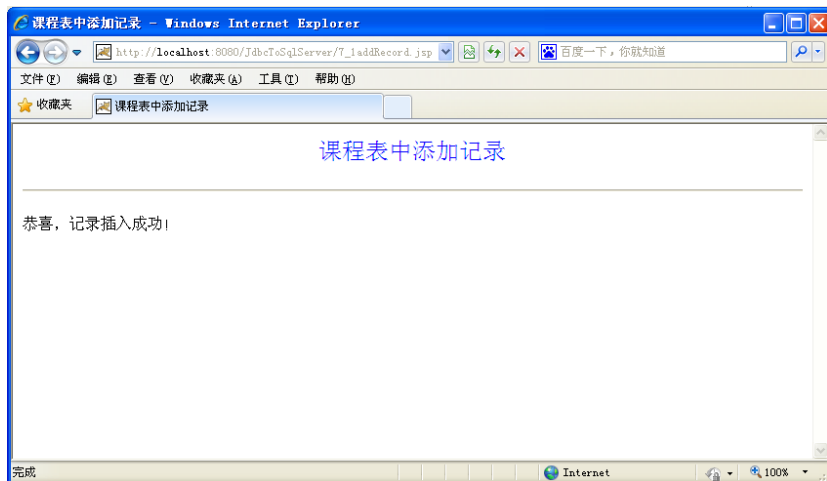


图 7.23 记录插入成功

执行完 7_1addRecord.jsp 后，读者可以通过 SQL Server Management Studio 查看该课程记录是否已经插到课程表中了，如图 7.24 所示，果然！课程表中多了一条“微机原理”课程的新记录（已在图 7.24 中框出）。

EASYBOOK-B2... J - dbo.KCB					
	KCH	KCM	KXXQ	XS	XF
▶	101	计算机基础	1	64	3
	102	C++程序设计	2	68	4
	103	数据结构	3	68	5
	104	计算机组成	3	96	4
	105	操作系统	4	80	5
	106	数据库原理	7	112	5
	107	计算机网络	5	96	4
	108	计算机新技术	1	32	2
	109	微机原理	1	80	5
	201	高等数学	1	80	5
	202	离散数学	3	68	4
*	NULL	NULL	NULL	NULL	NULL

图 7.24 多了一条课程新记录

7.4.2 查询记录

1. 查询语句 SELECT

对于数据库中记录的查询都是通过标准的 SQL 语句来实现的。SQL 的查询语句即 SELECT 语句，是 SQL 语言的核心，故这里用一定的篇幅专门讲解一下。

语法形式如下:

```
SELECT [DISTINCT] [别名.]字段名或表达式 [AS 列标题] FROM 表或视图 别名  
[ WHERE 条件]  
[ GROUP BY 分组表达式]  
[ ORDER BY 排序表达式[ ASC | DESC ]]
```

其中:

- SELECT 子句指出查询结果中显示的字段名或字段名和函数组成的表达式等, [AS 列标题]指定查询结果显示的列标题。若要显示表中所有字段, 可用通配符“*”代替字段名列表。可用 DISTINCT 去除重复的记录行。
- FROM 子句指定表或视图。
- WHERE 子句定义了查询条件。
- GROUP BY 子句对查询结果分组。
- ORDER BY 子句对查询结果排序。

例如:

- ① 查询 XSCJ 数据库的 XSB 表中各个同学的姓名和总学分:

```
USE XSCJ  
SELECT XM, ZXF FROM XSB
```

- ② 查询 XSB 表中各个同学的所有信息:

```
SELECT * FROM XSB
```

- ③ 查询 XSB 表中总学分大于或等于 50 的同学的情况:

```
SELECT * FROM XSB  
WHERE ZXF>=50
```

- ④ 查询 XSB 表中所在系为“计算机”, 且总学分大于或等于 50 的同学的情况:

```
SELECT * FROM XSB  
WHERE ZY='计算机' AND ZXF>=50
```

- ⑤ 查询 XSB 表中姓“王”的学生情况:

```
SELECT * FROM XSB  
WHERE XM LIKE '王%'
```

- ⑥ 查询 XSB 表中不在 1991 年出生的学生情况:

```
SELECT * FROM XSB  
WHERE CSRQ NOT BETWEEN '1991-1-1' AND '1991-12-31'
```

- ⑦ 查询总学分尚不定的学生情况:

```
SELECT * FROM XSB  
WHERE ZXF IS NULL
```

- ⑧ 查找计算机系学生的姓名、课程名和考试分数情况:

```
SELECT XM, KCM, CJ  
FROM XSB, KCB, CJB  
WHERE ZY='计算机' AND XSB.XH= CJB.XH AND KCB.KCH = CJB.KCH
```

- ⑨ 查找选修了课程号为 101 的课程的学生情况:

```
SELECT * FROM XSB  
WHERE XH IN  
( SELECT XH FROM CJB WHERE KCH = '101' )
```


在执行包含子查询的 SELECT 语句时，系统先执行子查询，产生一个结果表，再执行外查询。本例中，先执行子查询“SELECT XH FROM CJB WHERE KCH = '101'”。

⑩ 查找课程号 103 的成绩不低于课程号 101 的最低成绩的学生的学号：

```
SELECT XH FROM CJB
WHERE KCH = '103' AND CJ !< ANY
  ( SELECT CJ FROM CJB WHERE KCH= '101' )
```

这是比较子查询，可以认为这种子查询是 IN 子查询的扩展，它使表达式的值与子查询的结果进行比较运算。

⑪ 查找选修 103 号课程的学生姓名：

```
SELECT XM FROM XSB
WHERE EXISTS
  ( SELECT * FROM CJB WHERE XH= XSB.XH AND KCH = '103' )
```

EXISTS 谓词用于测试子查询的结果是否为空表，若子查询的结果集不为空，则 EXISTS 返回 TRUE，否则返回 FALSE。EXISTS 还可与 NOT 结合使用，即 NOT EXISTS，其返回值与 EXISTS 刚好相反。

⑫ 查找选修了全部课程的同学的姓名（即查找没有一门功课不选修的学生）：

```
SELECT XM FROM XSB
WHERE NOT EXISTS
  ( SELECT * FROM KCB
    WHERE NOT EXISTS
      ( SELECT * FROM CJB
        WHERE XH=XSB.XH AND KCH=KCB.KCH )
  )
```

⑬ 查询各课程，成绩按学号分组：

```
SELECT XH, CJ FROM CJB
GROUP BY XH
```

⑭ 将计算机系的学生按出生日期先后排序：

```
SELECT * FROM XSB
WHERE ZY= '计算机'
ORDER BY CSRQ
```

可以使用聚合函数对数据库表进行查询。常用的聚合函数如表 7.4 所示。

表 7.4 聚合函数表

函 数 名	说 明
AVG	求组中值的平均值
COUNT	求组中项数，返回 int 类型整数
MAX	求最大值
MIN	求最小值
SUM	返回表达式中所有值的和
VAR	返回给定表达式中所有值的统计方差

例如:

(1) 求选修 101 课程的学生的平均成绩:

```
SELECT AVG(CJ) AS '课程 101 平均成绩'
FROM CJB
WHERE KCH= '101'
```

(2) 求选修 101 课程的学生的最高分和最低分:

```
SELECT MAX(CJ) AS '课程 101 最高分', MIN(CJ) AS '课程 101 最低分'
FROM CJB
WHERE KCH= '101'
```

(3) 求学生的总人数:

```
SELECT COUNT(*) AS '学生总数'
FROM XSB
```

2. 普通查询

【例 7.2】 查询课程表的所有记录。

这里利用 `SQLServerConnBean` 的 `executeQuery()` 方法, 在 JSP 页面中利用 `SELECT` 语句向 `KCB` (课程表) 查询记录。

编写 JSP 程序 7_2select.jsp:

```
<%@ page contentType="text/html; charset=gb2312"%>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="SelectBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<head>
    <title>查询课程表的所有记录</title>
</head>
<body>
    <center>
        <font size="5" color="blue">查询课程表的所有记录</font>
    </center><br><hr><br>
    <table border="2" bordercolor="#ffcccc" align="center">
        <tr bgcolor="#cccccc" align="center">
            <td>课程号</td>
            <td>课程名</td>
            <td>学期</td>
            <td>学时</td>
            <td>学分</td>
        </tr>
        <%
            String sql="select * from KCB";
            //调用 SelectBean 中加载 JDBC 驱动的成员函数
            SelectBean.OpenConn();
            ResultSet rs=SelectBean.executeQuery(sql);    //创建结果集
            while(rs.next())
            {
                %>
            <tr>
```

```

        <td><%=rs.getString("KCH")%></td>
        <td><%=rs.getString("KCM")%></td>
        <td><%=rs.getInt("KXXQ")%></td>
        <td><%=rs.getInt("XS")%></td>
        <td><%=rs.getInt("XF")%></td>
    </tr>
    <%
    }
    %>
<%
    out.print("课程表查询结果如下: ");
    rs.close();
    SelectBean.closeStmt();
    SelectBean.closeConn();
    %>
</table>
</body>
</html>

```

运行结果如图 7.25 所示。



图 7.25 查询课程表的所有记录

从显示的记录来看，【例 7.1】插入的记录（'109','微机原理',1,80,5）此时也已经出现在了课程表中。

3. 查询结果的分页显示技术

在实际应用中，如果从数据库中查询得到的记录特别多，甚至超过了屏幕的显示范围，这时就需要将结果分页显示。

【例 7.3】从 SQL Server 2008 的数据库 XSCJ 的 XSB（学生表）中查出所有的记录并全部显示。要求：采用分页技术，每页显示 8 条记录。

(1) 编写 JSP 程序

编写 7_3page.jsp 文件，实现分页功能，具体代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page language="java" import="java.sql.*" errorPage=""%>
<jsp:useBean id="PageBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<head>
    <meta http-equiv="content-type" content="text/html;charset=gb2312">
    <title>分页显示记录</title>
    <style type="text/css">
    <!--
        .style2
        {
            color:#ff2222;
            font-size:24px;
        }
    -->
    </style>
</head>
<body>
    <div align="center">
        <span class="style2">分页显示记录</span><br><hr><br>
    </div>
    <table border="2" bordercolor="#ffcccc" align="center">
        <tr align="center" >
            <td>学号</td>
            <td>姓名</td>
            <td>性别</td>
            <td>出生日期</td>
            <td>专业</td>
            <td>总学分</td>
            <td>备注</td>
        </tr>
    <%
        int intPageSize;           //一页显示的记录数
        int intRowCount;           //记录总数
        int intPageCount;          //总页数
        int intPage;               //待显示页码
        int i;
        intPageSize=8;             //设置一页显示的记录数
        String strPage;
        strPage=request.getParameter("page"); //取得待显示的页码
        if(strPage==null)
        {
            //表明在 QueryString 中没有 page 这一参数，此时显示第一页的数据
            intPage=1;
```



```

<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
    <title>精确查询示例</title>
</head>
<body>
    <font size="5" color="blue">学生信息查询</font><br>
    <form action="7_4selectName.jsp" method="post">
        <h4>
            输入姓名: <input type="text" name="name">
            <input type="submit" value="提交">
        </h4>
    </form>
    <form action="7_4selectScore.jsp" method="post">
        <h4>根据学分查询:<br>
            学分在    <input type="text" name="scoremin" value="0" size="4"> 和
            <input type="text" name="scoremax" value="100" size="7"> 之间
            <input type="submit" value="提交">
        </h4>
    </form>
</body>
</html>

```

② 编写 7_4selectName.jsp 文件，用来显示按姓名查询的结果集：

```

<%@ page contentType="text/html;charset=gb2312" %>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="SelectNameBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<body>
    <%
        String name=request.getParameter("name");    //获取提交的姓名
        if(name==null)
        {
            name="";
        }
        byte b[]=name.getBytes("ISO-8859-1");
        name=new String(b);
        String sql="select * from XSB where XM='"+name+"'";
        SelectNameBean.OpenConn();
        ResultSet rs=SelectNameBean.executeQuery(sql);
        if (rs.next()==false)
        {
            out.println("查无此人，请");
        }
    %>
    <a href="7_4select.html">返回</a>
    <%
    }

```



```

        else
        {
            ResultSet rs1=SelectNameBean.executeQuery(sql);
        }
    }
    %>
    <center>
        输出姓名为<font color="blue"><%=name%></font>的学生信息: <br>
        <table border="2" bordercolor="#ffcccc" align="center">
            <tr align="center">
                <td>学号</td>
                <td>姓名</td>
                <td>性别</td>
                <td>出生日期</td>
                <td>专业</td>
                <td>总学分</td>
                <td>备注</td>
            </tr>
        </table>
    <%
        while(rs1.next())
        {
    %>
        <tr>
            <td><%=rs1.getString("XH")%></td>
            <td><%=rs1.getString("XM")%></td>
            <td><%=rs1.getString("XB")%></td>
            <td><%=rs1.getDate("CSRQ")%></td>
            <td><%=rs1.getString("ZY")%></td>
            <td><%=rs1.getInt("ZXF")%></td>
            <td><%=rs1.getString("BZ")%></td>
        </tr>
    <%
        }
        rs1.close();
    %>
    </table>
    <br><a href="7_4select.html">返回</a>
</center>
<%
    }
    %>
    %>
    rs.close();
    SelectNameBean.closeStmt();
    SelectNameBean.closeConn();
    %>
</body>
</html>

```

③ 编写 7_4selectScore.jsp 文件，用来显示按学分查询的结果集：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="SelectScoreBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<body>
    <%
        //获取提交的学分的最大、最小值：
        String scoremax=request.getParameter("scoremax");
        if(scoremax==null)
        {
            scoremax="100";
        }
        String scoremin=request.getParameter("scoremin");
        if(scoremin==null)
        {
            scoremin="0";
        }
        String cond="ZXF <= "+scoremax+" and "+ "ZXF >= "+scoremin;
        String cond1="总学分在"+scoremin+"到"+scoremax;
        String sql="select * from XSB where "+cond;
        SelectScoreBean.OpenConn();
        ResultSet rs=SelectScoreBean.executeQuery(sql);
    %>
    <center>
        输出<font color="blue"><%=cond1%></font>范围内的学生信息: <br>
        <table border="2" bordercolor="#ffcccc" align="center">
            <tr align="center">
                <td>学号</td>
                <td>姓名</td>
                <td>性别</td>
                <td>出生日期</td>
                <td>专业</td>
                <td>总学分</td>
                <td>备注</td>
            </tr>
        <%
            while(rs.next())
            {
        %>
            <tr>
                <td><%=rs.getString("XH")%></td>
                <td><%=rs.getString("XM")%></td>
                <td><%=rs.getString("XB")%></td>
                <td><%=rs.getDate("CSRQ")%></td>
                <td><%=rs.getString("ZY")%></td>
```

```

        <td><%=rs.getInt("ZXF")%></td>
        <td><%=rs.getString("BZ")%></td>
    </tr>

<%
    }
%>
</table>
<br><a href="7_4select.html">返回</a>
</center>
<%
    rs.close();
    SelectScoreBean.closeStmt();
    SelectScoreBean.closeConn();
%>
</body>
</html>

```

(2) 运行程序

① 先运行 7_4select.html 文件，运行结果如图 7.27 所示。



图 7.27 条件查询页

② 如果想查找姓名为“王林”的学生，在图 7.27 的“输入姓名”一栏中输入该学生的姓名，然后单击“提交”按钮，结果如图 7.28 所示。

可见名叫“王林”的学生一共有 2 人，都查找出来了。

如果输入的学生不存在，则显示如图 7.29 所示的提示页。

③ 如果想查找总学分在 50~100 之间的学生，在图 7.27 中填写学分范围，然后单击“提交”按钮，就可列出指定学分范围内所有学生的信息，如图 7.30 所示。

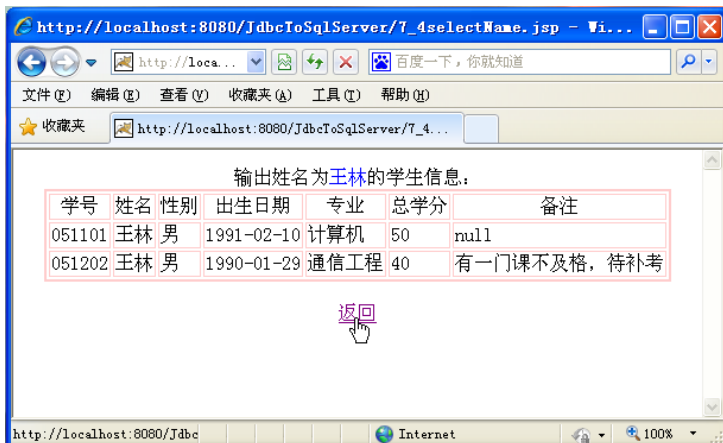


图 7.28 显示姓名为“王林”的学生信息

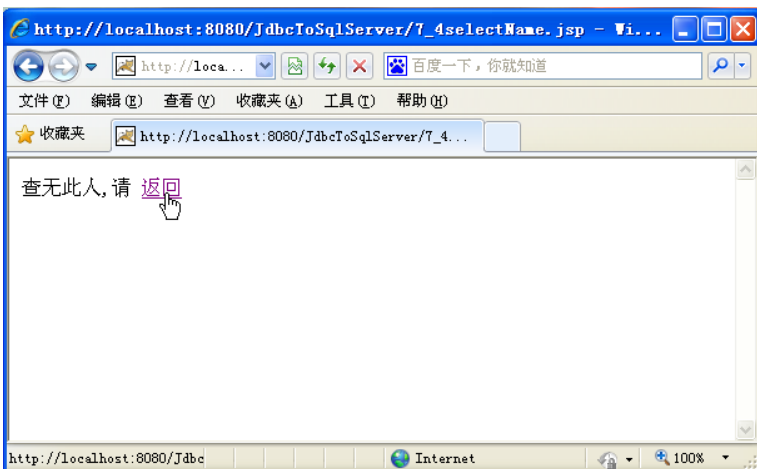


图 7.29 提示“查无此人”



图 7.30 显示学分在 50~100 之间的学生信息

(3) 代码分析

① 源文件 7_4select.html 建立了两个表单：一个用于按姓名查询（表单处理程序是 7_4selectName.jsp）；另一个则用于按学分查询（表单处理程序是 7_4selectScore.jsp）。

② 按姓名查询时，首先使用 request 对象的 getParameter()方法取得输入姓名的字符串 name，然后使用 getBytes("ISO-8859-1")进行汉字编码的转换，否则，表单传递的姓名变量值会出错。具体代码如下：

```
String name=request.getParameter("name");
if(name==null)
{
    name="";
}
byte b[]=name.getBytes("ISO-8859-1");
name=new String(b);
```

③ 按学分查询时，注意组合查询条件的写法，主体结构 with 简单查询相同，WHERE 子句由各条件组合而成，如：

```
String cond="ZXF <= "+scoremax+" and "+ZXF >= "+scoremin;
```

④ 按姓名查询，添加代码如下：

```
String sql="select * from XSB where XM = '"+name+"'";
SelectNameBean.OpenConn();
ResultSet rs=SelectNameBean.executeQuery(sql);
```

⑤ 按学分查询，添加代码如下：

```
String cond="ZXF <= "+scoremax+" and "+ZXF >= "+scoremin;
String sql="select * from XSB where "+cond;
SelectScoreBean.OpenConn();
ResultSet rs=SelectScoreBean.executeQuery(sql);
```

7.4.3 更新记录

UPDATE 语句用来更新表中的记录。

语法格式：

```
UPDATE 表名 SET 字段名 1=值 [,字段名 2=值 ...][WHERE 条件]
```

例如，将计算机系的学生总学分增加 2：

```
UPDATE XSB SET ZXF = ZXF+2
WHERE ZY = '计算机'
```

【例 7.5】在 JSP 页面中用 UPDATE 语句修改 KCB（课程表）中课程的学分值。

编写 JSP 程序 7_5select.jsp：

```
<%@ page contentType="text/html; charset=gb2312"%>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="SelectBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<head>
    <title>更新课程表</title>
</head>
<body>
```

```

<font size="5" color="blue">更新课程表</font><br><hr><br>
<form action="7_5update.jsp" method="post">
    输入课程号: <input type="text" name="kch"><br>
    输入新的学分: <input type="text" name="xf" size="10">
    <input type="submit" value="提交更新">
</form>
<p>数据库更新前记录: </p>
<table border="2" bordercolor="#ffcccc">
    <tr bgcolor="cccccc" align="center">
        <td>课程号</td>
        <td>课程名</td>
        <td>学期</td>
        <td>学时</td>
        <td>学分</td>
    </tr>
    <%
        String sql="select * from KCB";
        SelectBean.OpenConn();
        ResultSet rs=SelectBean.executeQuery(sql);
        while(rs.next())
        {
    %>
    <tr>
        <td><%=rs.getString("KCH")%></td>
        <td><%=rs.getString("KCM")%></td>
        <td><%=rs.getInt("KXXQ")%></td>
        <td><%=rs.getInt("XS")%></td>
        <td><%=rs.getInt("XF")%></td>
    </tr>
    <%
        }
    %>
    <%
        rs.close();
        SelectBean.closeStmt();
        SelectBean.closeConn();
    %>
</table>
</body>
</html>

```

编写 JSP 程序 7_5update.jsp:

```

<%@ page contentType="text/html; charset=gb2312" %>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="UpdateBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<body>

```

```

<%
    String KCH=request.getParameter("kch");    //获取课程号
    if(KCH==null)
    {
        KCH="";
    }
    String XF=request.getParameter("xf");        //获取学分
    if(XF==null)
    {
        XF="-1";
    }
    String sql="update KCB set XF = "+XF+" where KCH='"+KCH+"'";
    UpdateBean.OpenConn();
    UpdateBean.executeUpdate(sql);
%>
<p>更新后数据库记录: </p>
<table border="2" bordercolor="#ffcccc">
    <tr bgcolor="cccccc" align="center">
        <td>课程号</td>
        <td>课程名</td>
        <td>学期</td>
        <td>学时</td>
        <td>学分</td>
    </tr>
%>
    UpdateBean.OpenConn();
    ResultSet rs=UpdateBean.executeQuery("select * from KCB");
    while(rs.next())
    {
%>
        <tr>
            <td><%=rs.getString("KCH")%></td>
            <td><%=rs.getString("KCM")%></td>
            <td><%=rs.getInt("KXXQ")%></td>
            <td><%=rs.getInt("XS")%></td>
            <td><%=rs.getInt("XF")%></td>
        </tr>
%>
    }
%>
    rs.close();
    UpdateBean.closeStmt();
    UpdateBean.closeConn();
%>
</table>

```

```
</body>
```

```
</html>
```

先运行 7_5select.jsp, 结果如图 7.31 所示。



图 7.31 更新前的课程表

在 7_5select.jsp 页面中输入 KCB 表中已有的课程号 (如 “101”), 将学分由 “3” 修改为 “4”, 然后单击 “提交更新” 按钮, 执行 7_5update.jsp 中的代码。运行结果如图 7.32 所示。

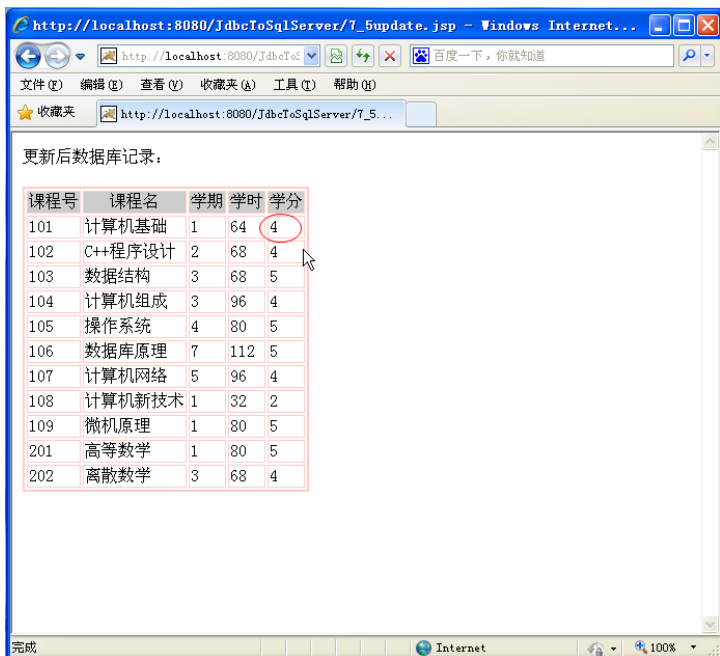


图 7.32 更新后的课程表

这时，可以看到课程号“101”的学分已经由原来的“3”变为“4”了（图 7.31 和图 7.32 中椭圆圈出的部分），说明数据库更新成功！

7.4.4 删除记录

DELETE 用来从表中删除记录。

语法格式：

```
DELETE FROM 表名 [WHERE 条件]
```

例如，从 XSB 表中删除姓名为“林时”的记录：

```
DELETE FROM XSB WHERE XM = '林时'
```

【例 7.6】在 JSP 页面中利用 DELETE 语句实现 KCB（课程表）记录的删除。要求：先显示 KCB 表中已有的记录，执行完删除操作后，再显示更新后的记录。

编写程序 7_6select.jsp：

```
<%@ page contentType="text/html; charset=gb2312"%>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="SelectBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<head>
    <title>更新课程表</title>
</head>
<body>
    <font size="5" color="blue">更新课程表</font><br><hr><br>
    <form action="7_6delete.jsp" method="post">
        输入要删除的课程号： <input type="text" name="kch" size="11">
        <input type="submit" value="提交">
    </form>
    <p>数据库更新前记录： </p>
    <table border="2" bordercolor="#ffcccc">
        <tr bgcolor="cccccc" align="center">
            <td>课程号</td>
            <td>课程名</td>
            <td>学期</td>
            <td>学时</td>
            <td>学分</td>
        </tr>
        <%
            String sql="select * from KCB";
            SelectBean.OpenConn();
            ResultSet rs=SelectBean.executeQuery(sql);
            while(rs.next())
            {
                %>
                <tr>
                    <td><%=rs.getString("KCH")%></td>
                    <td><%=rs.getString("KCM")%></td>
```

```

        <td><%=rs.getInt("KXXQ")%></td>
        <td><%=rs.getInt("XS")%></td>
        <td><%=rs.getInt("XF")%></td>
    </tr>
    <%
    }
    %>
    <%
    rs.close();
    SelectBean.closeStmt();
    SelectBean.closeConn();
    %>
</table>
</body>
</html>

```

编写程序 7_6delete.jsp:

```

<%@ page contentType="text/html;charset=gb2312" %>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="DeleteBean" scope="page" class="xscj_bean.SQLServerConnBean" />
<html>
<body>
    <%
    String KCH=request.getParameter("kch");
    if(KCH==null)
    {
        KCH="";
    }
    String sql="DELETE FROM KCB WHERE KCH='"+KCH+"'";
    DeleteBean.OpenConn();
    DeleteBean.executeUpdate(sql);
    %>
    <p>删除后数据库的记录: </p>
    <table border="2" bordercolor="#ffcccc">
        <tr bgcolor="cccccc" align="center">
            <td>课程号</td>
            <td>课程名</td>
            <td>学期</td>
            <td>学时</td>
            <td>学分</td>
        </tr>
        <%
        DeleteBean.OpenConn();
        ResultSet rs=DeleteBean.executeQuery("SELECT * FROM KCB");
        while(rs.next())
        {
            %>

```

```

        <tr>
            <td><%=rs.getString("KCH")%></td>
            <td><%=rs.getString("KCM")%></td>
            <td><%=rs.getInt("KXXQ")%></td>
            <td><%=rs.getInt("XS")%></td>
            <td><%=rs.getInt("XF")%></td>

        </tr>
    <%
    }
    %>
<%
    rs.close();
    DeleteBean.closeStmt();
    DeleteBean.closeConn();

    %>
</table>
</body>
</html>

```

先运行 7_6select.jsp 文件，运行结果如图 7.33 所示。

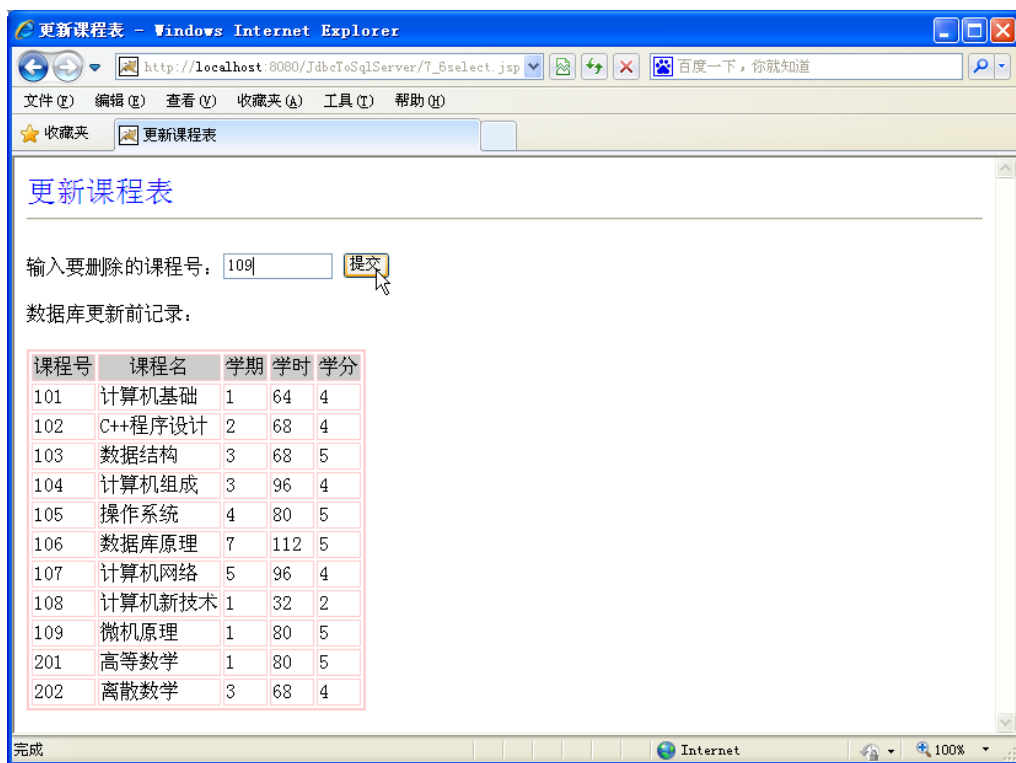


图 7.33 删除前的课程表

在图 7.33 所示的页面中输入要删除课程的课程号（如“109”），然后单击“提交”按钮，执行 7_6delete.jsp 中的代码。

进行删除操作之后的新课程表如图 7.34 所示。

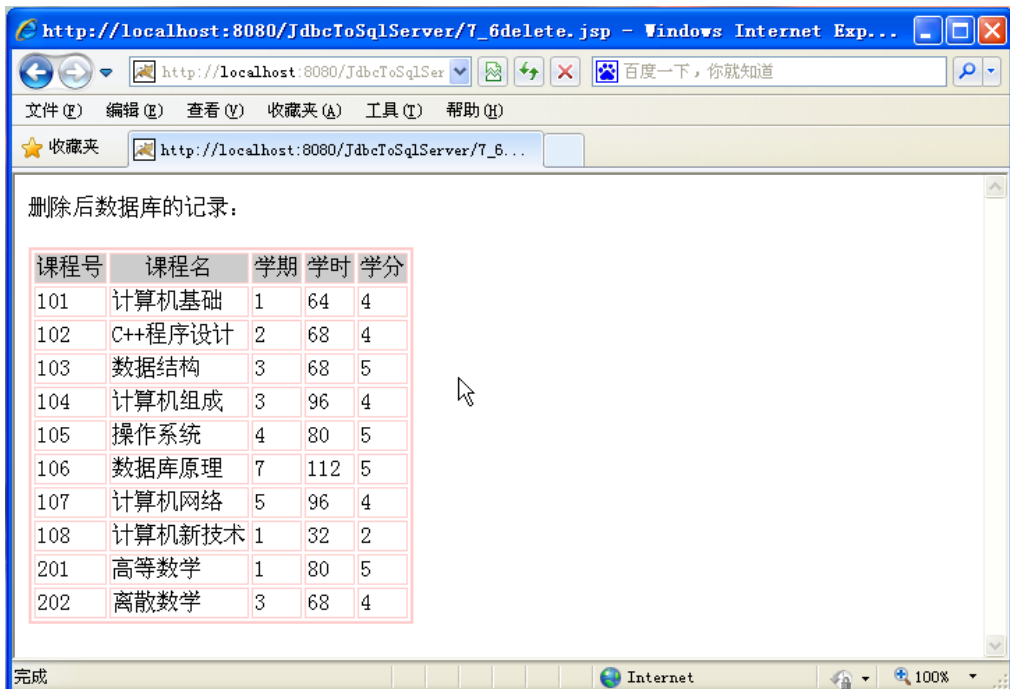


图 7.34 删除后的课程表

这时，可以看到课程号为“109”的课程（微机原理）已经被删除。

7.4.5 使用存储过程

存储过程在概念上类似于程序中的函数，它们获取输入参数，以黑盒模式运行并返回相应的信息。与函数不同的是，存储过程由数据库引擎执行，而不是在程序中执行。由于存储过程执行速度快，而且可以在系统启动时自动执行，不必在系统启动后再进行手工操作，大大方便了用户的使用。本节以 SQL Server 2008 数据库为基础，通过一个实例来说明在 JSP 中怎样实现存储过程。

1. 创建存储过程

通过 Management Studio 环境定义一个存储过程，实现向 XSCJ 数据库的 KCB（课程表）添加课程记录的功能，步骤如下。

(1) 在 Management Studio 的对象资源管理器窗口中，依次展开 XSCJ 数据库的子目录树下的“可编程性”→“存储过程”，单击鼠标右键，弹出如图 7.35 所示的快捷菜单，选择“新建存储过程”菜单项。

(2) 在工作区窗口中输入定义存储过程的代码。这里定义一个名为 KC_Insert 的存储过程，用来实现向 KCB 表中插入记录，具体代码如下：

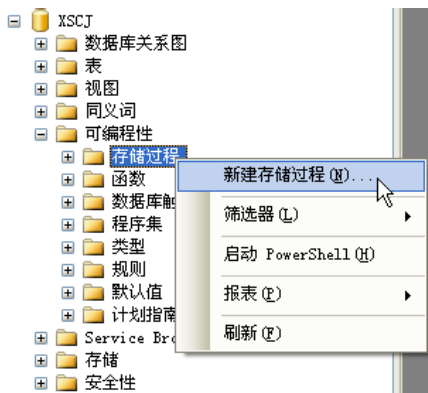


图 7.35 新建存储过程

```

CREATE PROCEDURE KC_Insert
@kch char(3),@kcm char(16),@kxxq tinyint,
@xs tinyint,@xf tinyint
AS
begin
    insert into KCB( KCH,KCM,KXXQ,XS,XF )
        values( @kch, @kcm,@kxxq,@xs,@xf)
end
GO

```

编辑完成后，执行上段代码，执行过后会发现 XSCJ 目录树的“存储过程”下多了一个子项“dbo. KC_Insert”，如图 7.36 的框出部分所示。

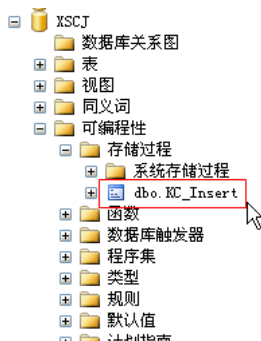


图 7.36 存储过程创建成功

至此，存储过程创建成功。

2. 编写 JSP 程序

编写 CallProce.jsp 文件，实现在 JSP 中调用 KC_Insert 存储过程，用来接收 JDBC 传来的参数。新增一门课程（JSP 基础）到 KCB 表中，具体代码如下：

```

<%@ page contentType="text/html; charset=gb2312" %>
<%@ page import="java.sql.*"%>
<%
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
    String url="jdbc:sqlserver://localhost:1433;databaseName=XSCJ";
    String user="sa";
    String password="123456";
    Connection conn= DriverManager.getConnection(url,user,password);
    CallableStatement stmt= conn.prepareCall("{ call KC_Insert (?,?,?,?) }" );    //调用存储过程
    stmt.setString(1,"303");
    stmt.setString(2,"JSP 基础");
    stmt.setInt(3,5);
    stmt.setInt(4,60);
    stmt.setInt(5,4);
    stmt.executeUpdate();
    stmt.close();
    conn.close();
    out.println("存储过程执行成功！");
%>

```

运行结果如图 7.37 所示。

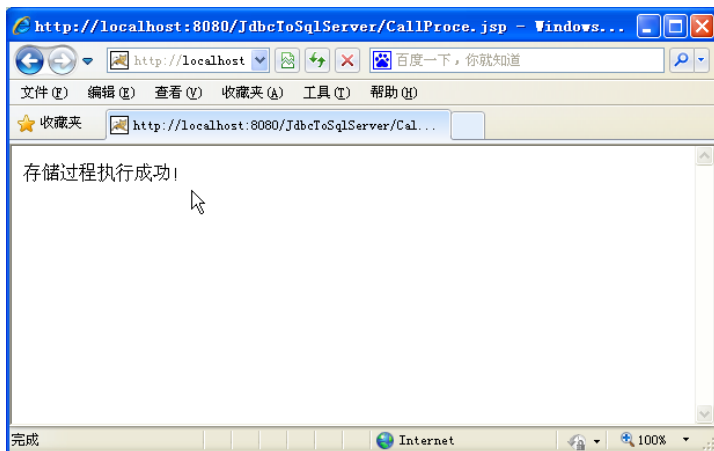


图 7.37 执行存储过程的结果

3. 代码分析

(1) 使用 JDBC 驱动访问数据库，实现与数据库交互。CallableStatement 接口用于执行数据库中的存储过程。有些存储过程要求用户输入参数，这时可以在生成的 CallableStatement 对象的存储过程调用语句中用问号设置输入的参数，然后在这个存储过程被执行之前使用 setXXX 方法给参数赋值，例如：

```
stmt.setString(1,"303");
```

另外，某些存储过程可能会返回输出参数，这时在执行这个存储过程之前，使用 registerOutParameter() 方法登记输出参数。要在 registerOutParameter() 方法中给出输出参数的相应位置及输出参数的 SQL 数据类型。如果想获得输出参数的值，可以使用 getXXX 方法，用法和 setXXX 方法相似。

本例通过以下语句来创建一个调用存储过程的语句：

```
CallableStatement stmt= conn.prepareCall("{ call KC_Insert (?,?,?,?) }");
```

(2) 创建了 CallableStatement 对象，就可以设置它的参数了。在 SQL Server 中创建存储过程时，使用 “@inparam1, @inparam2” 来表示存储过程中要使用的参数，在这里使用 stmt.setString(1,XXX) 之类的语句来设置它的参数值。

(3) 通过 stmt.executeUpdate() 来执行存储过程。执行这个语句后，数据就被保存在数据库中了，打开数据库，会发现课程表中多了一条记录，如图 7.38 的框出部分所示。

EASTBOOK-B2...J - dbo.KCB				
KCH	KCM	KXXQ	XS	XF
101	计算机基础	1	64	4
102	C++程序设计	2	68	4
103	数据结构	3	68	5
104	计算机组成	3	96	4
105	操作系统	4	80	5
106	数据库原理	7	112	5
107	计算机网络	5	96	4
108	计算机新技术	1	32	2
201	高等数学	1	80	5
202	离散数学	3	68	4
303	JSP基础	5	60	4
NULL	NULL	NULL	NULL	NULL

图 7.38 增加了“JSP 基础”课程记录

7.5 与其他数据库的互操作

除了 SQL Server 系列数据库，大家在日常办公中还会用到微软 Office 系列的 Access 数据库及 Excel 电子表格软件，因为这两种数据库没有提供 JDBC 驱动，所以 JSP 访问它们只能通过 JDBC-ODBC 桥，使用 ODBC 驱动实现访问。

7.5.1 连接 Access 2007

1. 创建一个 Access 型数据库

创建一个 xs.mdb 数据库，然后在数据库中创建一个 XSB 数据表，这个数据表包括 XH(学号)、XM(姓名)、XB(性别)、NL(年龄)和 ZY(专业)五个字段，并设置 XH 为主键。XSB 数据表中各字段属性如图 7.39 所示。

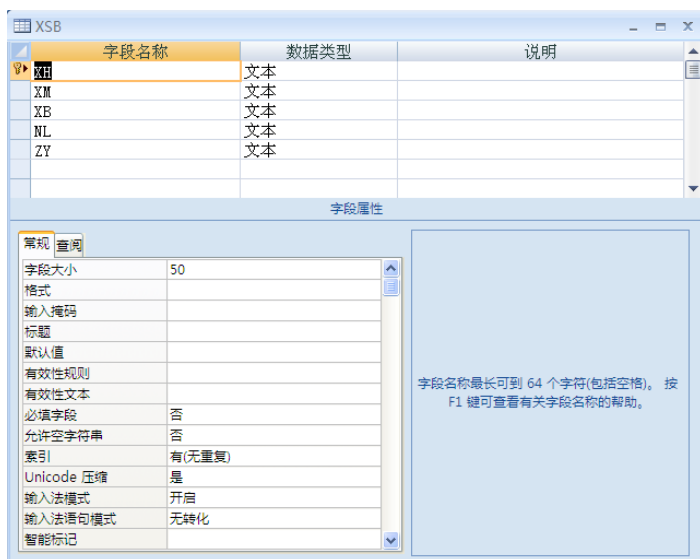


图 7.39 XSB 数据表中各字段属性

在数据表中输入学生的信息，数据表内容如图 7.40 所示。

XH	XM	XB	NL	ZY
06011201	贺芳芳	F	22	计算机科学与技术
06011202	张海	M	23	计算机科学与技术
06011203	李娜	F	21	计算机科学与技术
06011204	李霞	F	24	计算机科学与技术
06011205	李宏	M	21	计算机科学与技术
06011206	何勇	F	25	计算机科学与技术
06011207	范世杰	F	24	计算机科学与技术
06011208	黄晓晓	F	23	计算机科学与技术

图 7.40 数据表内容

2. 创建 ODBC 数据源

要通过 JDBC-ODBC 桥访问数据库，就必须先为数据库建立一个 ODBC 数据源，这样数据库才能实现和应用程序的交互。下面为 xs.mdb 数据库创建一个数据源，具体操作步骤如下。

(1) 选择“开始”→“控制面板”→“管理工具”→“数据源(ODBC)选项”菜单项，打开“ODBC 数据源管理器”对话框，如图 7.41 所示。



图 7.41 “ODBC 数据源管理器”对话框

(2) 单击“添加”按钮，打开“创建新数据源”对话框，如图 7.42 所示。

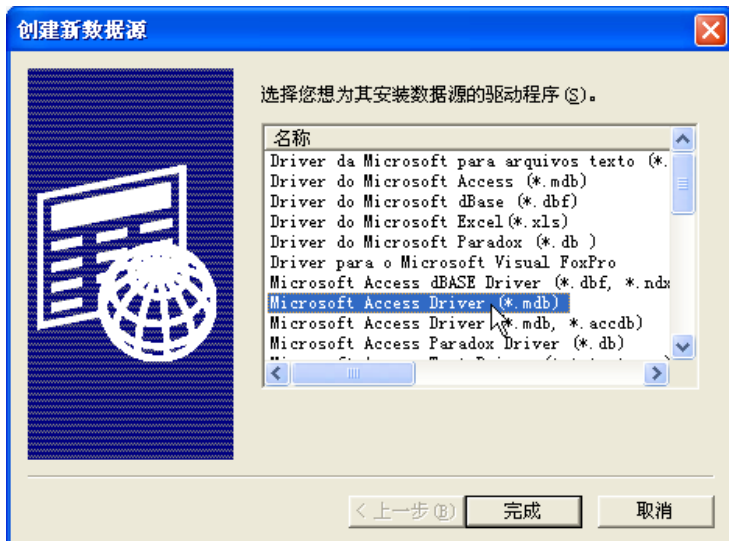


图 7.42 “创建新数据源”对话框

(3) 选中“Microsoft Access Driver(*.mdb)”选项，然后单击“完成”按钮，打开“ODBC Microsoft Access 安装”对话框，如图 7.43 所示。



图 7.43 “ODBC Microsoft Access 安装”对话框

(4) 在“数据源名”文本框中输入要创建的数据源名称，本例输入数据源名为“xs_access”，然后单击“数据库”框内的“选择”按钮，打开“选择数据库”对话框，如图 7.44 所示。

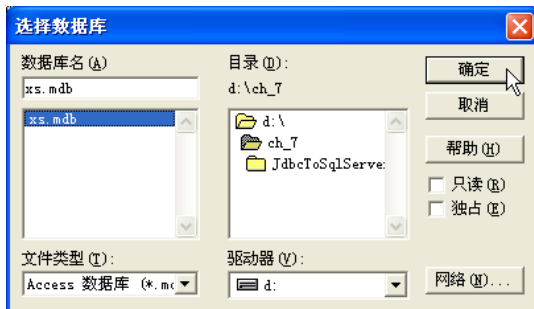


图 7.44 “选择数据库”对话框

(5) 选择前面创建的 xs.mdb 数据库，然后单击“确定”按钮，回到图 7.41 所示的窗口中，可以看到刚才创建的“xs_access”数据源已经出现在此窗口中（见图 7.45 中的圈出部分），再单击“确定”按钮，数据源创建过程结束。

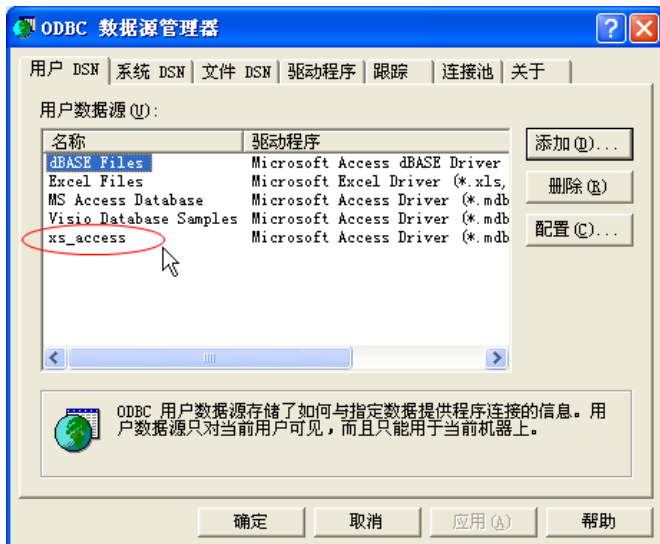


图 7.45 数据源创建成功

3. 测试可用性

启动 MyEclipse, 创建 Web 项目, 将工程命名为 “JOdbcToAccess”。

(1) 创建操作数据库的 JavaBean

在 src 目录下创建包 xscj_bean, 在包中创建类 AccessConnBean, 该类实现了一个操作数据库的 JavaBean。

代码如下:

```
package xscj_bean;
import java.sql.*;
public class AccessConnBean {
    private Statement stmt=null;
    private Connection conn = null;
    ResultSet rs=null;
    //构造函数
    public AccessConnBean() { }
    public void OpenConn()throws Exception
    {
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url="jdbc:odbc:xs_access";
            String user="";
            String pwd="";
            conn=DriverManager.getConnection(url,user,pwd);
        }
        catch(SQLException e)
        {
            System.err.println("Data.executeQuery: " +e.getMessage());
        }
    }
    //执行查询类的 SQL 语句, 有返回集
    public ResultSet executeQuery(String sql) {
        rs = null;
        try
        {
            stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE
                                           ,ResultSet.CONCUR_READ_ONLY);
            rs = stmt.executeQuery(sql);
        }
        catch(SQLException e)
        {
            System.err.println("Data.executeQuery: " + e.getMessage());
        }
        return rs;
    }
    //关闭对象
```

```

    public void closeStmt(){
        try
        {
            stmt.close();
        }
        catch(SQLException e)
        {
            System.err.println("Data.executeQuery: " + e.getMessage());
        }
    }
    public void closeConn(){
        try
        {
            conn.close();
        }
        catch(SQLException e)
        {
            System.err.println("Data.executeQuery: " + e.getMessage());
        }
    }
}

```

(2) 编写 JSP 程序

将 AccessConnBean 创建好之后, 编写 Access.jsp 文件, 该文件是通过 AccessConnBean 访问 Access 数据库的 JSP 程序。

代码如下:

```

<%@ page contentType="text/html;charset=gb2312"%>
<%@ page language="java" import="java.sql.*"%>
<jsp:useBean id="AccessBean" scope="page" class="xscj_bean.AccessConnBean" />
<html>
<head>
    <title>使用 JDBC-ODBC 桥访问 Access 数据库</title>
</head>
<body>
    <center>
        <font size="5" color="blue">JDBC-ODBC 桥访问 Access 数据库</font>
    </center><br><hr><br>
    <table border="2" bordercolor="#ffcccc" align="center">
        <tr bgcolor="cccccc" align="center">
            <td><b>记录序列号</b></td>
            <td><b>学号</b></td>
            <td><b>姓名</b></td>
            <td><b>性别</b></td>
            <td><b>年龄</b></td>
            <td><b>专业</b></td>
        </tr>

```

```

<%
    String sql="select * from XSB";           //查询 XSB 表中所有字段的记录
    AccessBean.OpenConn();
    ResultSet rs=AccessBean.executeQuery(sql); //创建结果集
    while(rs.next())
    {
    %>
    <tr align="center">
        <td><%=rs.getRow()%></td>
        <td><%=rs.getString("XH")%></td>
        <td><%=rs.getString("XM")%></td>
        <td><%=rs.getString("XB")%></td>
        <td><%=rs.getString("NL")%></td>
        <td><%=rs.getString("ZY")%></td>
    </tr>
    <%
    }
    %>
    <% out.print("数据库操作成功, 恭喜你! ");%>
    <%
        rs.close();
        AccessBean.closeStmt();
        AccessBean.closeConn();
    %>
</table>
</body>
</html>

```

运行结果如图 7.46 所示。



图 7.46 访问 Access 成功

4. 程序说明

上面的实例利用 JavaBean 通过 JDBC-ODBC 桥访问 Access 数据库，下面简单介绍一下。

(1) AccessConnBean.java 代码

- 加载驱动程序。在 JDBC 连接到 ODBC 数据库之前，必须加载 JDBC-ODBC 桥的驱动程序，代码为：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
```

该语句使用了 Class 类(java.lang 包)中的方法 forName 载入该驱动程序的类“sun.jdbc.odbc.JdbcOdbcDriver”，从而创建了该驱动程序的一个实例。

- 创建数据库连接。加载 JDBC-ODBC 桥的驱动程序后，就可以连接数据库了。首先创建 Connection 类的一个实例“conn”，并使用 DriverManager 对象的 getConnection() 来测试使用“url”指定的数据库连接。创建数据库连接的代码如下：

```
String url="jdbc:odbc:xsdata";
String user="";
String pwd="";
conn=DriverManager.getConnection(url,uesr,pwd);
```

这里的 java.sql.Connection 类用来管理 JDBC 和数据库之间的连接，它还提供了对 SQL 语言的支持，以便操纵数据库、进行事务处理。java.sql.DriverManager 是驱动程序管理器，其参数“url”用来指定连接的 ODBC 数据源，“user”和“pwd”用来指定访问数据库的用户名和密码。

- 访问数据库。使用 Connection 类对象的 createStatement()方法从指定的数据库得到一个 Statement 的实例“stmt”，然后使用这个实例的 executeQuery()方法来执行 SQL 语句，并将查询结果保存到 ResultSet 对象“rs”中。实现代码如下：

```
stmt =conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                ResultSet.CONCUR_READ_ONLY);

ResultSet rs = stmt.executeQuery(String sql);
```

- 关闭数据库连接，释放资源。对数据库的访问结束后，及时地关闭 ResultSet 对象、Statement 对象和 Connection 对象，从而释放所占的资源。实现代码如下：

```
rs.close();
stmt.close();
conn.close();
```

(2) JSP 页面的代码

获取数据库中的记录并输出。数据库查询的结果保存在 ResultSet 对象“rs”中。用于获取数据库中的记录并输出如下代码：

```
<%
...
ResultSet rs=AccessBean.executeQuery(sql);
while(rs.next())
{
%>
<tr align="center">
<td><%=rs.getRow()%></td>
<td><%=rs.getString("XH")%></td>
<td><%=rs.getString("XM")%></td>
```

```

<td><%=rs.getString("XB")%></td>
<td><%=rs.getString("NL")%></td>
<td><%=rs.getString("ZY")%></td>
</tr>
<%
}
%>

```

这里主要用到了 `rs.next()` 方法, `rs.getRow()` 方法和 `rs.getString()` 方法。其中 `rs.getRow()` 方法用来获取当前记录所在的行数, 即当前记录是第几条记录。其他的两个方法在 7.4.2 节中已经介绍过, 这里不再赘述。

7.5.2 连接 Excel 2007

由于 Excel 简单易学, 而且功能强大, 所以在实际生活中, 人们经常将一些重要的数据保存在 Excel 表格中。下面就介绍如何利用 JSP 查询 Excel 表格中的数据。

1. 创建一个 Excel 文件

首先创建一个 Excel 表格文件, 文件名为 `xs.xls`, 然后在其 `Sheet1` 中录入学生的基本信息, 包括 `XH` (学号)、`XM` (姓名)、`XB` (性别)、`NL` (年龄) 和 `ZY` (专业), 共五个字段。创建好的 `xs.xls` 如图 7.47 所示。

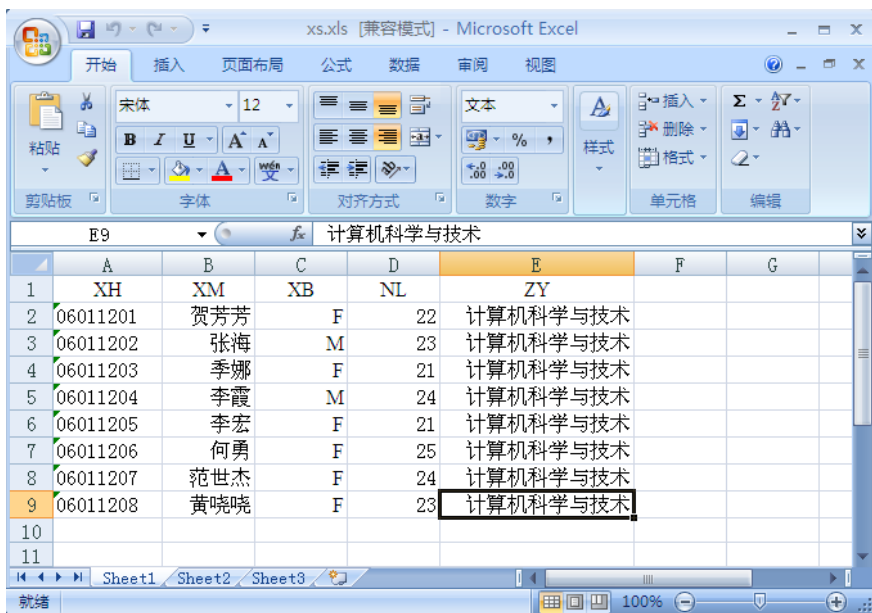


图 7.47 创建 Excel 表格

2. 创建 Excel 数据源

使用 JDBC-ODBC 桥来访问 Excel 表格中的数据, 需要创建 Excel 数据源。创建 Excel 数据源和创建 Access 数据源的方法非常相似, 只需在“创建新数据源”窗口中选择“Microsoft Excel Driver(*.xls)”选项, 然后在“ODBC Microsoft Excel 安装”窗口中输入数据源名“`xs_excel`”, 选择刚才创建的 `xs.xls` 文件即可。

3. 编写 JSP 程序

前面都是用 JavaBean 实现的与数据库的连接，这里在 JSP 页面中直接用代码实现与数据库的交互。启动 MyEclipse，创建 Web 项目，将工程命名为“JODbcToExcel”。

创建一个 Excel.jsp 文件，代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page language="java" import="java.sql.*" errorPage=""%>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=gb2312">
    <title>使用 JSP 查询 Excel 表格中的数据</title>
</head>
<body>
    <center>
        <font size="5" color="blue">使用 JSP 查询 Excel 表格中的数据</font>
    </center><br><hr><br>
    <table border="2" bordercolor="#ffcccc" align="center">
        <tr bgcolor="cccccc" align="center">
            <td width="80" align="center">学号</td>
            <td width="80" align="center">姓名</td>
            <td width="80" align="center">性别</td>
            <td width="80" align="center">年龄</td>
            <td width="80" align="center">专业</td>
        </tr>
        <%
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection conn=DriverManager.getConnection("jdbc:odbc:xs_excel","", "");
            Statement stmt=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE
                                                        ,ResultSet.CONCUR_UPDATABLE);

            String sql="select * from [Sheet1$]";           //从 xs.xls 数据表的 sheet1 中取数据
            ResultSet rs=stmt.executeQuery(sql);           //创建结果集
            while(rs.next())
            {
        %>
        <tr align="right">
            <td><%=rs.getString("XH")%></td>
            <td><%=rs.getString("XM")%></td>
            <td><%=rs.getString("XB")%></td>
            <td><%=rs.getString("NL")%></td>
            <td><%=rs.getString("ZY")%></td>
        </tr>
        <%
            }
        %>
        <% out.print("数据库操作成功，恭喜你！");%>
        <%
            rs.close();
```

```

        stmt.close();
        conn.close();
    %>
</table>
</body>
</html>

```

运行结果如图 7.48 所示。



图 7.48 访问 Excel 成功

4. 程序说明

使用 JDBC-ODBC 桥访问 Excel 表格中的数据和使用 JDBC-ODBC 桥访问 Access 数据库的实现方法非常相似，都先要创建 ODBC 数据源，通过数据源访问数据。

- 首先连接 Excel 的数据源，代码如下：

```

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn=DriverManager.getConnection("jdbc:odbc:xs_excel","", "");

```

数据源名称为 xs_excel；用户名和密码都为空。

- 连接 Excel 表后，就可以从表中查询相应的数据了，代码如下：

```

Statement stmt=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                       ResultSet.CONCUR_UPDATABLE);

String sql="select * from [Sheet1$]";
ResultSet rs=stmt.executeQuery(sql);

```

从上面的代码可以看出，从 Excel 表中查询数据和从 SQL Server、Access 数据库中查询数据一样，都是通过 SQL 语句实现的。其中 “[Sheet1\$]” 就相当于数据库中的表的名称。

7.6 上机练习

1. 按照 7.2 节的指导，安装和配置 SQL Server 2008 环境，建立数据库和表，并输入表 7.1～表 7.3 的数据。

2. 在 MyEclipse 中创建数据库连接, 并编程测试连接的可用性, 如图 7.49 所示 (与图 7.22 相同), 一旦能从数据库中读出学生信息列表, 就表示之前配置的连接是成功的。

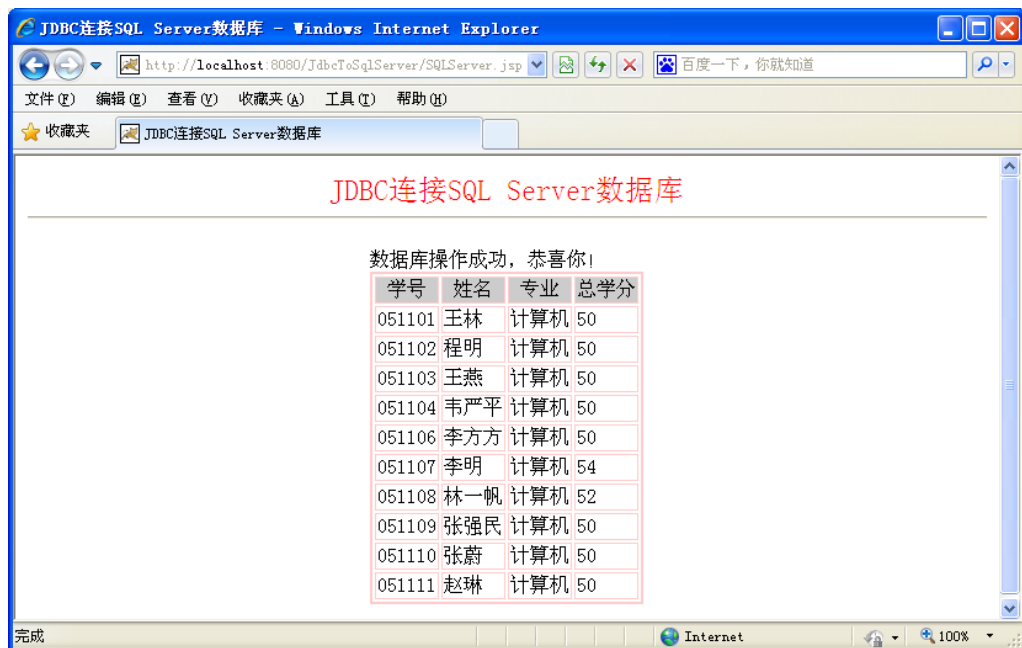


图 7.49 数据库连接成功

请读者自己练习用同样的方式编写代码, 访问并显示数据库中另外两个表 (课程表和成绩表) 的基本信息。

3. 在成功建立连接的基础上, 对照 7.4 节编程实现数据库的插入、查询、更新、删除以及使用存储过程等操作。

4. 尝试用 JSP 连接和操作 Access 2007、Excel 2007 等常用数据库。

通过前面各章节的学习，大家已经比较全面地掌握了 JSP 的基础知识，本章以学生信息管理系统为例介绍 JSP 的综合应用，包含学生信息检索、新生信息的录入和删除等功能。

8.1 JSP 系统的架构方式

8.1.1 表示层的两种架构模式

一个应用程序只有与用户进行交互才能够为用户提供服务。在一个 Web 应用中，表示层就是用户与应用程序之间沟通的纽带，它负责接收用户请求，传递给后台程序，并把后台程序处理的结果呈现给用户。在 Web 应用发展过程中，主要产生了两种表示层的架构模式：Model1 和 Model2。

1. Model1 架构模式

Model1 架构模式的工作原理如图 8.1 所示。

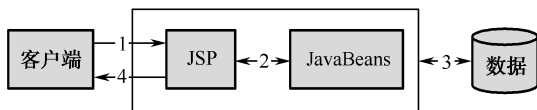


图 8.1 Model1 架构模式的工作原理图

如图 8.1 所示，Model1 架构模式的工作流程是按照如下四个步骤进行的。

- ① 客户端发出请求，该请求由 JSP 页面接收。
- ② JavaBean 用于实现业务逻辑，JSP 根据请求与不同 JavaBean 进行交互。
- ③ 业务逻辑操作数据库。
- ④ JSP 将结果信息转发给客户端。

Model1 架构模式的实现过程比较简单，在这种架构模式中，JSP 集控制和显示于一体。使用 Model1 架构模式能够快速开发出一些小型项目。但是在大型应用中，这种架构模式会为应用程序的开发设计带来负面影响。

- JSP 页面中既包含 HTML 标签，又包含 JavaScript 代码，同时还包含了大量 Java 代码，增加了 JSP 页面的维护难度及程序调试难度。

- 业务逻辑分布在各个 JSP 页面中，要想理解整个应用的执行流程，必须明白所有 JSP 页面的结构。
- 各个组件耦合紧密，修改某一业务逻辑或数据，需要同时修改多个相关页面。

2. Model2 架构模式

Model2 架构模式的工作原理如图 8.2 所示，其工作流程是按照如下五个步骤进行的。

- ① Servlet 接收客户端发出的请求。
- ② Servlet 根据不同的请求调用相应的 JavaBean。
- ③ 业务逻辑操作数据库。
- ④ Servlet 将结果传递给 JSP 视图。
- ⑤ JSP 将后台处理结果呈现给客户端。

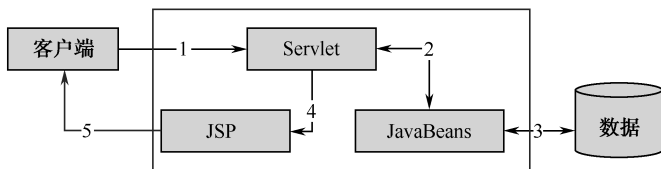


图 8.2 Model2 架构模式的工作原理图

与 Model1 架构模式相比，Model2 引入了 Servlet 组件，并将控制功能交由 Servlet 实现，而 JSP 将只负责显示功能。通过引入 Servlet，能够实现控制逻辑与显示逻辑的分离，从而提高了程序的可维护性。

8.1.2 MVC 基础

MVC 是一种交互界面的结构组织模型，使用 MVC 能够实现软件的计算模型与界面模型的分离；同时它也是一种通用的设计模式，不局限于某一种特定的编程语言及应用。MVC 设计模式对 Web 应用开发产生了深远影响，它使得开发人员分工更加明确，软件维护更加方便。

1. MVC 简史

MVC 设计模式最初只被应用于单机应用程序的开发设计。经典的 MVC 架构工作流程如图 8.3 所示。

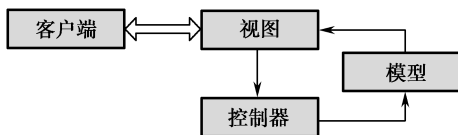


图 8.3 经典的 MVC 架构工作流程图

在基于这种 MVC 模型的应用中，用户直接与视图进行交互。用户在视图界面输入数据并单击相应按钮进行提交，控制器负责接收视图信息并对相应模型进行操作，根据用户提供的数据更新模型状态。模型状态发生变化后，控制器通知视图，视图根据模型的变化进行更新并显示给用户。

经典的 MVC 模型仅适用于单机应用程序的开发设计，在单机应用程序中，视图、模型及控制器都在同一台计算机中，实现起来非常容易。但是对于 Web 应用来说就不适用了，由于

Web 应用的特殊性, 视图在客户端, 而控制器和模型在服务器, 要想使用 MVC 模型就要改变其架构。于是, 在经典的 MVC 架构基础上又产生了两种其他架构方式: 前端控制器模式和页面控制器模式。这两种架构的 MVC 模型工作流程分别如图 8.4 和图 8.5 所示。

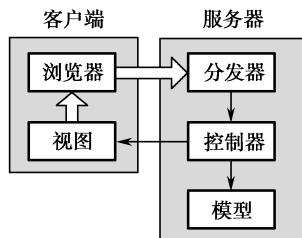


图 8.4 前端控制器模式

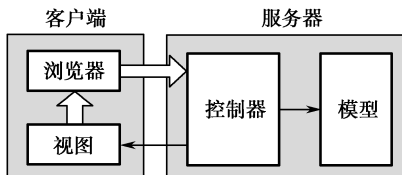


图 8.5 页面控制器模式

在前端控制器模式中, 应用程序引入了一个叫做分发器的组件 (某些 Web 应用中可以通过 Servlet 实现该功能)。分发器负责接收客户端浏览器发出的请求, 并根据请求的 URL 地址将信息转发给特定的控制器。控制器改变相应模型的状态并返回一个标志, 该标志与指定视图存在映射关系, 通过标志找到对应视图并在客户端浏览器显示执行结果。

页面控制器模式与前端控制器模式稍有不同, 它不是通过分发器去寻找指定的控制器, 而是在客户端浏览器中直接请求某个具体的控制器。页面控制器模式虽然造成了视图组件与控制器组件的耦合过于紧密, 但是在某些时候能够提高应用程序的执行效率。

2. MVC 的基本构成

MVC 设计模式将一个完整的应用分为三个组件: Model (模型)、View (视图) 及 Controller (控制器)。在实际应用中, 这三个组件既相互独立又相互关联。

- **Model (模型):** 该组件是对软件所处理问题逻辑的一种抽象, 封装了问题的核心数据、逻辑和功能实现, 独立于具体的界面显示及 I/O 操作。
- **View (视图):** 该组件负责将表示模型数据、逻辑关系及状态的信息, 以某种形式展现给用户。视图组件从模型组件获得显示信息, 并且对于相同的显示信息可以通过不同的显示形式或视图展现给用户。
- **Controller (控制器):** 该组件主要负责用户与软件之间的交互操作, 控制模型状态变化的传播, 以确保用户界面与模型状态的统一。Web 应用中, 当用户请求到来时, 控制器本身不输出任何信息也不做任何处理, 只是接收请求并决定调用哪个模型去处理该请求, 然后确定使用哪个视图组件来显示模型处理返回的数据。

基于 MVC 模型的 Web 应用的基本工作流程如图 8.6 所示。

图 8.6 的整个工作流程可以分为四个步骤。

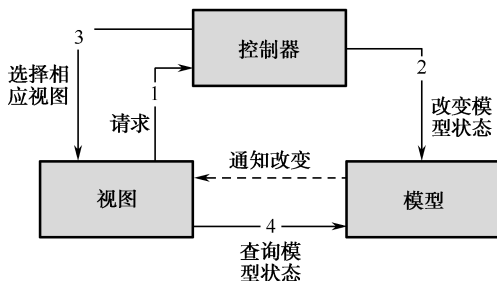


图 8.6 MVC 模型的基本工作流程图

- ① 用户通过视图（一般是 JSP 页面或 HTML 页面）发出请求。
- ② 控制器接收请求后，调用相应的模型并改变其状态。
- ③ 当模型状态改变后，控制器选择对应的视图组件来反馈改变后的结果。
- ④ 视图根据改变后的模型，将正确的状态信息显示给用户。

通常情况下，一个控制器只与一个视图相关联；但是，一个模型却可以与多个视图相关联。当某个模型状态发生变化时，所有与其相关联的视图都要更新，以保持视图信息与模型状态的一致性。

3. MVC 的优缺点

虽然 MVC 设计模式在面向对象程序设计中被广泛应用，但是该设计模式并不是十全十美的。我们必须了解并掌握 MVC 的优点及缺点，这样在实际开发过程中才能够扬长避短，完全发挥 MVC 设计模式的优势。

（1）MVC 的优点

MVC 的优点体现在如下三个方面。

- 有利于分工部署。使用 MVC 设计模式开发应用程序，不同职能的人员分工非常明确。以使用 Java 语言开发 Web 应用为例，Java 程序员只需将精力集中于业务逻辑，而界面程序员（HTML 和 JSP 开发人员）则将精力集中在页面的表现形式上。
- 降低耦合，提高可维护性。MVC 设计模式将表示层与业务层有效分离，降低了二者之间的耦合程度。这样一来，任何业务逻辑的变动都不会影响到表示层代码；同样，开发人员可以随意修改表示层代码，而不用重新编译模型和控制器的代码。
- 提高应用程序的重用性。对于同一个 Web 应用，客户可能会使用多种方式进行访问，如普通计算机的 IE 浏览器、智能手机的 WAP 浏览器等。但无论通过何种方式，应用程序的业务逻辑及其流程都是一样的，需要改变的只是表示层的具体实现方式。由于 MVC 实现了业务与视图的分离，故而能够轻松解决这一问题。

（2）MVC 的缺点

MVC 的缺点体现在如下两个方面。

- MVC 并不适合小型应用程序的开发设计。MVC 要求开发人员完全按照模型、视图及控制器三个组件的模式对应用程序进行划分。对于某些小型应用来说，反而会增加一些不必要的工作，影响开发效率。
- 基于 MVC 设计模式进行程序设计，要求开发人员在编程之前，必须精心设计程序的结构；在设计过程中，由于将一个完整的应用划分为三个组件，相应增加了需要管理的文件数量。

8.1.3 Struts 1 框架

为了简化基于 MVC 的 JSP 开发，很多开源组织自发地研制出一类称做“框架”的软件，并免费供给 JSP 程序员使用。Struts 1 是最早出现也是迄今为止最为流行的 MVC 框架，能够很好地帮助 JSP 程序员利用 MVC 思想去开发 Web 项目。

Struts 的工作流程如图 8.7 所示。

从图 8.7 中可以知道 Struts 的工作流程如下所述。

- ① 客户端发出请求，表单封装数据，然后提交给 Action Servlet。
- ② Action Servlet 根据请求信息找到指定的 Action，并将请求转发给 Action。

- ③ Action 调用 JavaBean（即模型）提供的业务逻辑方法处理请求，并返回 Action Forward。
- ④ Action Servlet 根据 Action Forward 信息，将请求转发给 JSP 页面（视图）。
- ⑤ 将最终页面返回给客户端。

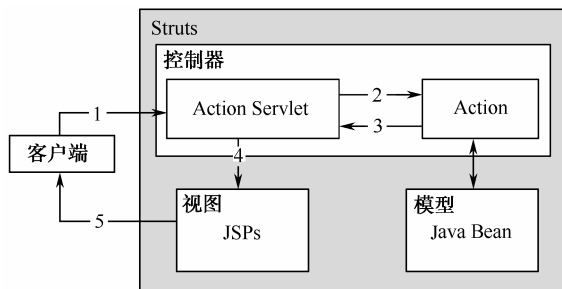


图 8.7 Struts 的工作流程图

可以看出，应用 Struts 框架符合 MVC 编程思想，条理也很清晰。尽管如此，Struts 框架的设计还存在很多问题，比如它的 Action 与 Servlet API 紧密地结合在一起，使程序的耦合度大大增加，这对开发大型的 Web 系统是十分不利的，不过对于像本章将要开发的学生信息管理系统这种规模不太大的应用来说，用 Struts 1 就足够了，且能借此机会熟悉 Servlet 的编程及其控制原理。

注意：在 JavaEE 领域使用 Struts 2 更为普遍，Struts 2 源自 WebWork，并不是 Struts 1 的直接升级版。故而 Struts 1 与 Struts 2 是两个完全不同的框架系列，当前它们都有自己的最新发布版本并在不同场合发挥着各自的作用。一般情况下所说的 Struts 指的就是 Struts 1 框架而非 Struts 2。

8.2 开发前的准备工作

本案例使用 MyEclipse 集成环境开发，由于是综合性的 JSP 应用，涉及的方方面面都需要在开发之前做好相关的准备。

8.2.1 创建 Web 工程

启动 MyEclipse，新建 Web Project，命名为“StudentInfo”。作为 JSP 的综合应用，工程目录的结构要稍复杂些，如图 8.8 所示。

请读者参照图 8.8，建立项目的目录结构。在此就图 8.8 的目录结构略做如下说明。

- ① src 目录存放 java 代码，其下有三个包。
 - org.model: 存放模型，主要为连接和操作数据库的组件。
 - org.util: 存放共享 java 类和代码，如系统程序公用的常量、方法的定义等。
 - org.web: 存放各个 Action 对应的 java 类的代码。



图 8.8 目录结构

② WebRoot 下存放的都是表示层的 JSP 代码、脚本及资源文件。

- **css 文件夹：**系统中各 JSP 页面通用的样式表定义。在该目录下创建文件 `userCN.css`，定义本项目用到的样式表：

```
.list_table{background-color:#fffff; border:#000000; border-style:solid; border-top-width:0px;
            border-right-width: 0px; border-bottom-width:0px; border-left-width:0px}
.list_table_tr_title{text-align:center; font-family:"Times New Roman","SimSun"; font-size:12px;
                    color:#003333; font-weight:normal; background-color:#dfe4e9; height:20px}
.list_table_tr_even{font-family:"Times New Roman","SimSun"; font-size:12px; color:#000000;
                    font-weight: normal; background-color: #FFFFFF; height:18px}
.list_table_tr_odd{font-family:"Times New Roman","SimSun"; font-size:12px; color:#333333;
                    font-weight: normal; background-color: #EFEFEF; height:18px}
```

- **images 文件夹：**存放想要在页面上显示的图片资源，请读者上网获取南京师范大学网站首页的标头图片 `njnu.jpg`（见图 8.9），并置于该目录下以备后用。



图 8.9 南京师范大学网站标头图片

- **includes 文件夹：**存放页面共通 JSP，这部分代码定义在 `taglib.jsp` 文件中，如下：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%
    response.setHeader("Cache-Control","no-store"); //HTTP 1.1
    response.setHeader("Pragma","no-cache"); //HTTP 1.0
    response.setDateHeader("Expires", 0); //prevents caching at the proxy server
%>
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=GBK">
    <link rel="stylesheet" href="../css/userCN.css">
    <meta http-equiv="Expires" content="0">
    <title>学生信息系统</title>
</head>
<script language="javascript" src="../script/check_value.js"></script>
```

其中，`struts-bean.tld`、`struts-html.tld` 和 `struts-logic.tld` 这三个 `tld` 文件是 Struts 库中的内容，后面会有说明。

- **script 文件夹：**放置各 JSP 页面通用的脚本代码。本例放的是判断用户输入表单项值是否为空的代码，在 `check_value.js` 文件中，如下：

```
function checkNULL(value)
{
    if(value == null || value == "")
    {
```

```
        return false;
    }
    return true;
}
```

- user 文件夹：这个文件夹下存放的是实现页面功能的 JSP 源文件，包括 user_list.jsp 和 user_insert.jsp 两个文件，它们分别实现“显示学生信息列表页”和“插入新生信息页”这两项最为重要的界面显示功能。
- ③ WebRoot 下的子目录 WEB-INF 中包括三个文件夹和一个 web.xml 文件，作用如下：
 - config 文件夹：存放 Struts 1 的 xml 配置文件，主要为配置项目中的各个组件模块（实现界面的 form-bean 及其 Action 映射）。本例包括两个配置文件（struts-config.xml 和 struts-config-user.xml），至于其中具体的配置内容，将在后面开发时介绍。
 - lib 文件夹：由 MyEclipse 自动创建，集中存放项目用到的第三方.jar 包和类库，Struts 库和 Jdbc 驱动就存放在其中。
 - tld 文件夹：放置 struts-bean.tld、struts-html.tld 和 struts-logic.tld 这三个 tld 文件，它们都是从 Struts 库中解压提取的。
 - web.xml 文件：与所有 Web 项目一样用来集中定义项目的配置信息，其中包含了对过滤器和 tld 文件的配置，后面会详细介绍。

8.2.2 导入 Struts 库

1. 下载 Struts 1

登录 <http://struts.apache.org/> 下载 Struts 1 完整版，如图 8.10 所示，本书使用的是 2008 年 12 月发布的 Struts 1.3.10。

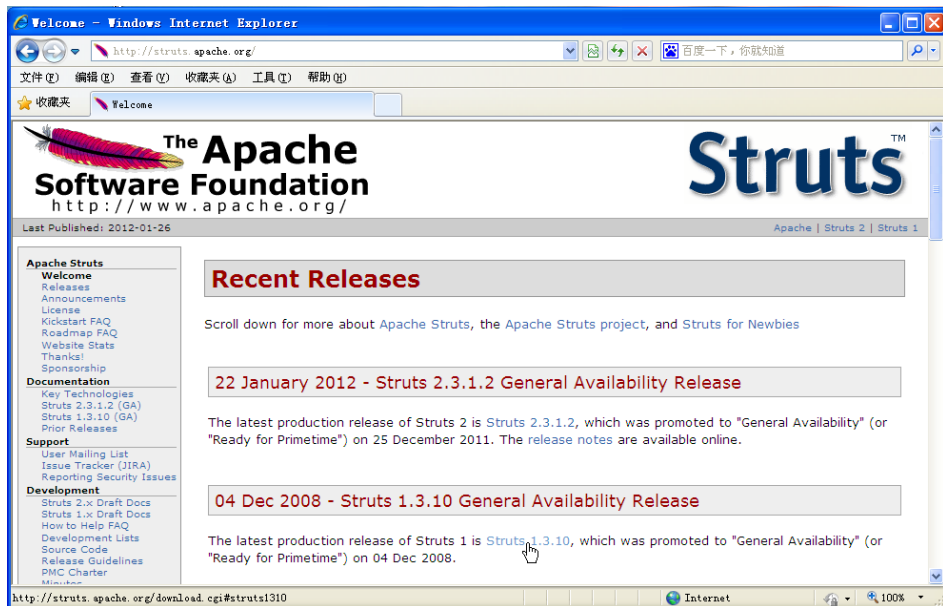


图 8.10 Struts 1 下载页

2. 导入 Struts 1 库

将下载的 zip 文件解压缩, 在其 lib 目录下包含了 Struts 1 所有的开发包, 如图 8.11 所示。

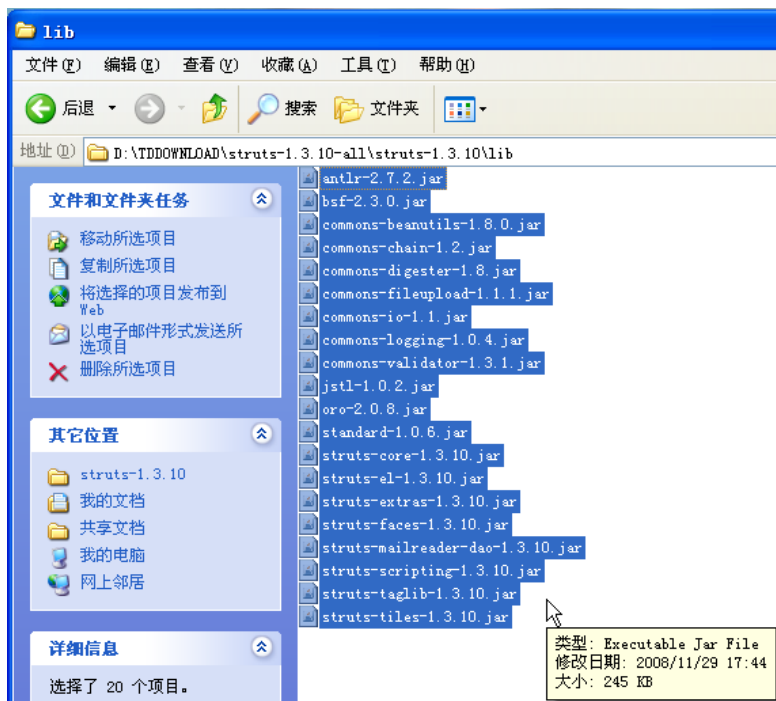


图 8.11 Struts 1 所有的开发包

一共有 20 个 jar 文件, 将它们复制到项目的\WebRoot\WEB-INF\lib 目录下, 然后右击项目名, 选择“Build Path”→“Configure Build Path”菜单项, 弹出如图 8.12 所示的对话框。

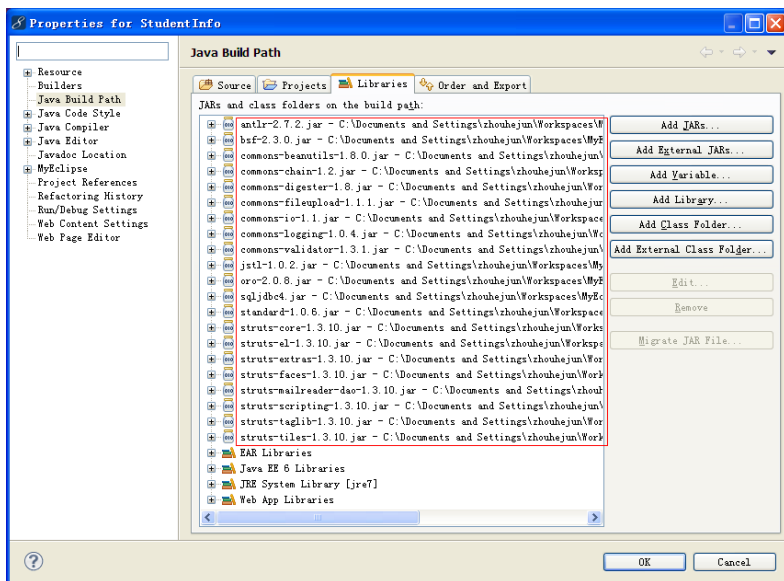


图 8.12 添加 Struts 1 的 jar 包

在“Libraries”选项卡中单击“Add External JARs”按钮，进入\WebRoot\WEB-INF\lib 目录，选中上面的 20 个 jar 包，单击“OK”按钮完成类库的添加。

3. 提取 tld 文件

tld 是标签库描述文件 (taglib description) 的缩写，如要在 JSP 页面中实现 JSP 标签，必须首先定义实现标签的类，然后在标签库描述文件 (tld) 中将写好的类映射成 jsp 标签，最后在 JSP 文件中使用定义好的标签，就可以生成动态的 JSP 内容。

在 Struts 1 开发中有三个 tld 文件是我们要用的，它们就是之前多次提及的 struts-bean.tld、struts-html.tld 和 struts-logic.tld，位于 struts-taglib-1.3.10.jar\META-INF\tld 下（需要用 WinRAR 一类的软件打开查看），如图 8.13 中的圈出部分所示。



图 8.13 找到 tld 文件

将这三个文件复制到项目的 WebRoot\WEB-INF\tld 目录下，然后在 web.xml 中进行配置：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app version="3.0"
```

```
  xmlns="http://java.sun.com/xml/ns/javaee"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
```

```
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
```

```
  ...
```

```
  <jsp-config>
```

```
    <taglib>
```

```
      <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
```

```
      <taglib-location>/WEB-INF/tld/struts-bean.tld</taglib-location>
```

```
    </taglib>
```

```
    <taglib>
```

```
      <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
```

```
      <taglib-location>/WEB-INF/tld/struts-html.tld</taglib-location>
```

```
    </taglib>
```

```
    <taglib>
```

```
      <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
```

```
      <taglib-location>/WEB-INF/tld/struts-logic.tld</taglib-location>
```

```
    </taglib>
```

```
  </jsp-config>
```

```
</web-app>
```

至此，Struts 1 库及其中的标签库就都可以在编程中使用了！

8.2.3 数据准备

本章学生信息管理系统的开发，在数据准备上依旧使用第 7 章表 7.1 的学生表（XSB，见图 8.14）及 SQL Server 2008 服务器。

EASTBOOK-B2...J - dbo.XSB						
XH	XM	XB	CSRQ	ZY	ZXF	BZ
051101	王林	男	1991-02-10	计算机	50	NULL
051102	程明	男	1992-02-01	计算机	50	NULL
051103	王燕	女	1990-10-06	计算机	50	NULL
051104	韦严平	男	1991-08-26	计算机	50	NULL
051106	李方方	男	1991-11-20	计算机	50	NULL
051107	李明	男	1991-05-01	计算机	54	提前修完《数...
051108	林一帆	男	1990-08-05	计算机	52	已提前修完一...
051109	张强民	男	1989-08-11	计算机	50	NULL
051110	张蔚	女	1992-07-22	计算机	50	三好生
051111	赵琳	女	1991-03-18	计算机	50	NULL
051113	严红	女	1990-08-11	计算机	48	有一门课不及...
051201	王敏	男	1989-06-10	通信工程	42	NULL
051202	王林	男	1990-01-29	通信工程	40	有一门课不及...
051203	王玉民	男	1991-03-26	通信工程	42	NULL
051204	马琳琳	女	1989-02-10	通信工程	42	NULL
051206	李计	男	1990-09-20	通信工程	42	NULL
051210	李红庆	男	1990-05-01	通信工程	44	已提前修完一...
051216	孙祥欣	男	1989-03-09	通信工程	42	NULL
051218	孙研	男	1991-10-09	通信工程	42	NULL
051220	吴薇华	女	1991-03-18	通信工程	42	NULL
051221	刘燕敏	女	1990-11-12	通信工程	42	NULL
051241	罗琳琳	女	1991-01-30	通信工程	50	转专业学习
*	NULL	NULL	NULL	NULL	NULL	NULL

图 8.14 XSB 的数据

为了能够实现验证登录的功能，本项目还要增加一张学生登录表（XSDLB），用于存储每个学号的学生对应的登录口令，如图 8.15 所示。

EASTBOOK-B2...J - dbo.XSDLB	
XH	PASSWORD
051101	20061216
051102	123
051103	123
051104	123
051106	123
051107	123
051108	123
051109	123
051110	123
051111	123
051113	123
051201	123
051202	123
051203	123
051204	123
051206	123
051210	123
051216	123
051218	123
051220	123
051221	123
051241	123
*	NULL

图 8.15 XSDLB 的数据

准备好数据之后,要记得将 SQL Server 2008 的 Jdbc 驱动包 sqljdbc4.jar 也导入项目工程中,操作方法与导入 Struts 1 库相同。

8.2.4 配置过滤器

Struts 1 开发要求程序员自己手工编写和配置过滤器,本节讲有关过滤器的基础知识,并为本章的实例配置过滤器。

1. 过滤器简介

过滤器是在 Java Servlet 2.3 规范中定义的,它能够对 Servlet 容器传给 Web 组件的 ServletRequest 对象和 ServletResponse 对象进行检查和修改。过滤器本身并不生成 ServletRequest 对象和 ServletResponse 对象,它只为 Web 组件提供如下过滤功能。

① 过滤器能够在 Web 组件被调用之前检查 ServletRequest 对象,修改请求头和请求正文的内容,或者对请求进行预处理操作。

② 过滤器能够在 Web 组件被调用之后检查 ServletResponse 对象,修改响应头和响应正文。

③ 过滤器负责过滤的 Web 组件可以是 Servlet、JSP 或 HTML 文件。

过滤器的过滤过程如图 8.16 所示。

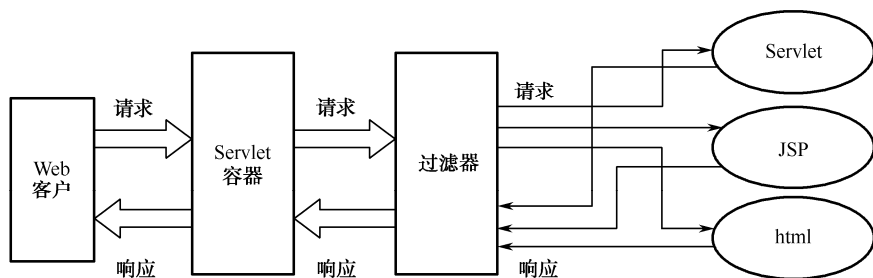


图 8.16 过滤器的过滤过程

过滤器具有以下特点。

- 过滤器可以检查 ServletRequest 和 ServletResponse 对象,并且利用 ServletRequestWrapper 和 ServletResponseWrapper 包装类来修改 ServletRequest 和 ServletResponse 对象。
- 可以在 web.xml 文件中为过滤器映射特定的 URL。当客户请求访问此 URL 时,Servlet 容器就会先触发过滤器工作。
- 过滤器是 Java Servlet 2.3 规范的一部分,因此所有实现 Java Servlet 2.3 规范及其以上版本的 Servlet 容器都支持过滤器。
- 多个过滤器可以被串联在一起,协同为 Web 组件过滤请求对象和响应对象。

2. 创建过滤器

所有自定义的过滤器类都必须实现 javax.servlet.Filter 接口,这个接口含有以下三个过滤器类必须实现的方法。

- `init(FilterConfig config)`: 这是过滤器的初始化方法。在 Web 应用启动时,Servlet 容器先创建包含了过滤器配置信息的 FilterConfig 对象,然后创建 Filter 对象,接着调用 Filter 对象的 `init(FilterConfig config)` 方法,在这个方法中,可通过 config 参数来读取 web.xml 文件中为过滤器配置的初始化参数。

- **doFilter(ServletRequest req, ServletResponse res, FilterChain chain):** 这个方法完成实际的过滤操作。当客户请求访问的 URL 与为过滤器映射的 URL 匹配时, Servlet 容器将先调用过滤器的 doFilter()方法。FilterChain 参数用于访问后续过滤器或 Web 组件。
- **destroy():** Servlet 容器在销毁过滤器对象前调用该方法, 在这个方法中可以释放过滤器占用的资源。

过滤器由 Servlet 容器创建, 在它的生命周期中包含以下初始化阶段。

- **加载阶段:** 当 Web 应用启动时, Servlet 容器会加载过滤器类, 创建过滤器配置对象 (FilterConfig) 和过滤器对象, 并调用过滤器对象的 init(FilterConfig config)方法。
- **运行时阶段:** 当客户请求访问的 URL 与为过滤器映射的 URL 匹配时, Servlet 容器将先调用过滤器的 doFilter()方法。
- **销毁阶段:** 当 Web 应用终止时, Servlet 容器将先调用过滤器对象的 destroy()方法, 然后销毁过滤器对象。

下面来为学生信息管理系统编写过滤器。

在 src\org\util 下创建 Java 类文件 UrlEncodingFilter.java, 编辑代码如下:

```
package org.util;
import java.io.*;
import javax.servlet.*;

public class UrlEncodingFilter implements Filter
{
    protected String encoding = null;
    protected FilterConfig filterConfig = null;
    protected boolean ignore = true;
    public void init(FilterConfig filterConfig) throws ServletException
    {
        this.filterConfig = filterConfig;
        this.encoding = filterConfig.getInitParameter("encoding");
        String value = filterConfig.getInitParameter("ignore");
        if (value == null)
        {
            this.ignore = true;
        }
        else if (value.equalsIgnoreCase("true"))
        {
            this.ignore = true;
        }
        else if (value.equalsIgnoreCase("yes"))
        {
            this.ignore = true;
        }
        else
        {
            this.ignore = false;
        }
    }
}
```

```

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException
{
    if (ignore || (request.getCharacterEncoding() == null))
    {
        String encoding = selectEncoding(request);
        if (encoding != null)
        {
            request.setCharacterEncoding(encoding);
        }
    }
    chain.doFilter(request, response);
}

protected String selectEncoding(ServletRequest request)
{
    return (this.encoding);
}

public void destroy()
{
    this.encoding = null;
    this.filterConfig = null;
}
}

```

在 `UrlEncodingFilter` 的 `init(FilterConfig filterConfig)` 初始化方法中, 先调用 `filterConfig.getInitParameter("encoding")` 方法, 从 `web.xml` 文件中读取初始化参数 `encoding`, 接着调用 `filterConfig.getInitParameter("ignore")` 方法, 从 `web.xml` 文件中读取初始化参数 `ignore`。在 `UrlEncodingFilter` 的 `doFilter()` 方法中, 先后调用 `selectEncoding()` 和 `setCharacterEncoding()` 方法, 对客户请求进行预处理。

3. 发布过滤器

在发布过滤器时, 必须在 `web.xml` 文件中加入 `<filter>` 元素和 `<filter-mapping>` 元素。`<filter>` 元素用来定义一个过滤器, 本例的配置代码如下所示:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
<filter>
    <filter-name>UrlEncodingSet</filter-name>
    <filter-class>org.util.UrlEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>GB2312</param-value>
    </init-param>
    <init-param>
        <param-name>ignore</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>

```

```
<filter-mapping>
    <filter-name>UrlEncodingSet</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

...

```
</web-app>
```

在以上代码中，`<filter-name>`子元素指定过滤器的名字，`<filter-class>`指定过滤器的类名。`<init-param>`子元素为过滤器实例提供初始化参数，它包含一对参数名和参数值，在`<filter>`元素中可以包含多个`<init-param>`子元素。在这里定义了一个名为“encoding”的参数，以及一个名为“ignore”的参数。在`UrlEncodingFilter`类的`init(FilterConfig filterConfig)`方法中，可通过`filterConfig.getInitParameter(String name)`来读取初始化参数。

`<filter-mapping>`元素用于为过滤器映射特定的 URL，对于以上代码，当客户请求的 URL 和`<url-pattern>`指定的 URL(/*)匹配时，Servlet 容器就会先调用`UrlEncodingFilter`过滤器的`doFilter()`方法。`<filter-mapping>`元素中的`<filter-name>`子元素必须和`<filter>`元素中的`<filter-name>`子元素一致。

8.3 模型组件的开发

在学生信息管理系统中，作为后台程序的模型（model）组件的主要职能是连接和操作数据库，包括 `DBConnection`、`UserBean` 和 `UserInfoManager` 三个类，源文件位于项目的 `src\org\model` 目录下。

8.3.1 连接数据库

在 `org.model` 包中创建 `DBConnection` 类，`DBConnection.java` 代码如下：

```
package org.model;
import java.sql.*;
public class DBConnection
{
    public static Connection getConnection() throws SQLException
    {
        Connection conn = null;
        try
        {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            String url = "jdbc:sqlserver://localhost:1433;databaseName=XSCJ";
            String user = "sa";
            String password = "123456";
            conn = DriverManager.getConnection(url,user,password);
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }
    }
}
```

```
        if (conn != null) {conn.close();}
    }
    return conn;
}
}
```

这是借助 Jdbc 访问的 SQL Server 数据库，DBConnection 类仅负责连接上数据库，至于对数据的操作则交给 UserManager 完成。

8.3.2 定义 POJO 类

POJO (Plain Old Java Objects) 是简单的 Java 对象，实际就是普通 JavaBeans，是这样的一种“纯粹的”JavaBean：在它里面除了 JavaBean 规范的方法和属性外没有别的东西，即只有 private 属性及对这个属性存取的 public 类型的 get 和 set 方法。

UserBean.java 代码如下：

```
package org.model;
public class UserBean
{
    // 学号
    private String studentId;
    // 姓名
    private String studentName;
    // 性别
    private String sex;
    // 生日
    private String birtyDate;
    // 专业
    private String subject;
    // 密码
    private String passWord;

    public String getStudentId()
    {
        return studentId;
    }
    public void setStudentId(String studentId)
    {
        this.studentId = studentId;
    }

    public String getStudentName()
    {
        return studentName;
    }
    public void setStudentName(String studentName)
    {

```



```
        this.studentName = studentName;
    }

    public String getSex()
    {
        return sex;
    }
    public void setSex(String sex)
    {
        this.sex = sex;
    }

    public String getBirtyDate()
    {
        return birtyDate;
    }
    public void setBirtyDate(String birtyDate)
    {
        this.birtyDate = birtyDate;
    }

    public String getSubject()
    {
        return subject;
    }
    public void setSubject(String subject)
    {
        this.subject = subject;
    }

    public String getPassWord()
    {
        return passWord;
    }
    public void setPassWord(String passWord)
    {
        this.passWord = passWord;
    }
}
```

大家会发现这样的 JavaBean 很“单纯”，它只能装载数据，作为数据存储的载体，而不具有业务逻辑处理的能力，但对操作数据来说是必不可少的，只有通过它，UserInformationManager 类才能够成功地操作后台的数据。

8.3.3 操作数据库

操作数据库的功能全部由 `UserInformationManager` 类实现，其代码的框架为：

```
package org.model;
import java.sql.*;
import org.util.UserConstants;
import java.util.ArrayList;
import org.web.UserInsertForm;
public class UserInformationManager
{
    private static UserInformationManager instance;
    private UserInformationManager(){}
    public static synchronized UserInformationManager getInstance()
    {
        if (instance == null)
        {
            instance = new UserInformationManager();
        }
        return instance;
    }
    /*判定输入用户的用户名和密码是否存在*/
    public boolean checkSubmitSuccess(String userId, String passWord)throws SQLException{...}
    /*查询学生信息 List */
    public ArrayList searchStudentList(String studentId, String studentName)throws SQLException{...}
    /*删除学生信息*/
    public void delStudentValue(String[] studentId) throws SQLException{...}
    /*根据学号检索学生 bean */
    public UserBean searchStudent(String studentId) throws SQLException{...}
    /*判定某学生的记录是否存在*/
    public boolean checkUserBack(String userId) throws SQLException{...}
    /*插入学生信息*/
    public void insertValue(UserInsertForm form) throws SQLException{...}
}
```

以上框架中所定义各个方法的具体实现细节，我们将在稍后结合该系统特定功能讲解时再分别单独地给出。

由于在系统运行的过程中需要多次访问同一个数据库表、执行相同的动作，将这些表和动作类型的名字定义为常量统一管理，有助于提高程序代码的可读性。在项目 `org.util` 包下创建类 `UserConstants.java`，定义如下：

```
package org.util;
public final class UserConstants
{
    // 单击登录的 actionType
    public static final String ACTION_SUBMIT = "submit";
    // 单击一览的 actionType
```

```
public static final String ACTION_LIST = "list";
// 单击插入的 actionType
public static final String ACTION_INSERT = "insert";
// 单击删除的 actionType
public static final String ACTION_DELETE = "delete";
// 判断错误信息
public static final String CHECK_ERROR = "error";
// 学生信息表表名
public static final String TAB_XSB = "XSB";
// 学生登录表表名
public static final String TAB_XSDL = "XSDLB";
}
```

这样一定义，在后面编程时就可以在代码中直接引用上述常量名了。

8.4 登录验证功能

作为一个完整的 JSP 应用系统，必须具有用户验证的能力，本案例识别用户合法性依据的是 XSDLB 中事先存储的“学号-密码”对。

8.4.1 登录页面设计

在 WebRoot 下新建 JSP 文件 logon.jsp:

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="org.util.*"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=GBK">
    <link rel="stylesheet" href="css/userCN.css">
    <meta http-equiv="Expires" Content="0">
    <title>学生信息系统</title>
</head>
<script language="javascript" src="script/check_value.js"></script>
<script language="javascript">
    if (top.location != this.location)
    {
        top.location = this.location;
    }
    function submitForm(act)
    {
        // 学号
        var userID = document.all.userId.value;
```

```

// 口令
var passWord = document.all.password.value;
// 判断用户名是否为空
if(!checkNULL(userID))
{
    alert("学号不可以为空!");
    return;
}
document.logonForm.actionType.value = act;
document.logonForm.submit();
}
function pressEnter()
{
    if(event.keyCode==13)
    {
        submitForm("submit");
    }
}
</script>
<body bgcolor="#bbbf" >
<html:form action="logon.do" method="post">
<html:hidden property="actionType"/>
<table width="680" border="0" cellspacing="1" cellpadding="0" align="center"
        bgcolor="#000000" height="35%">
<tr>
<td bgcolor="9eaccd" valign="bottom">
<table width="100%" border="0" align="center" cellpadding="0" cellspacing="0">
<tr>
<td height="1" bgcolor="#ffffff">
<div align="center"></div>
</td>
</tr>
</table>
<table width="100%" border="0" cellspacing="0" cellpadding="0"
        height="63" bgcolor="#ffffff">
<tr>
<td width="50%"></td>
<td height="29" width="50%">
<div align="center">
<p><font face='宋体' size="6">
<b>学生信息管理系统</b>
</font></p>
</div>
</td>
</tr>
</table>

```



```

        </tr>
      </table>
    </td>
  </tr>
</table>
<script language="javascript">
  document.all.userId.focus();
  <%
    String checkValue = (String)request.getAttribute("check");
    if(checkValue != null && UserConstants.CHECK_ERROR.equals(checkValue))
    {
  %>
  alert("对不起,你输入的学号或者口令有误!");
  <%
    }
  %>
</script>
</html:form>
</body>
</html>

```

在 web.xml 文件中设置该页面为项目的默认启动页:

```

<welcome-file-list>
  <welcome-file>logon.jsp</welcome-file>
</welcome-file-list>

```

设计的登录页面效果如图 8.17 所示。



图 8.17 登录页面

8.4.2 Action 功能模块

Struts 1 开发主要靠程序员编写 Action 来实现系统功能, 这里的 Action 相当于功能模块, 使得代码结构化程度高, 便于阅读、理解和维护。

在 org.web 包下创建 LogonAction.java:

```
package org.web;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.model.UserInformationManager;
import org.util.UserConstants;
public class LogonAction extends Action
{
    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                HttpServletRequest request, HttpServletResponse response)throws Exception
    {
        HttpSession session = request.getSession();
        DynaActionForm myform = (DynaActionForm) form;
        String actionType = (String) myform.get("actionType");
        // 用户名
        String userId = (String) myform.get("userId");
        // 口令
        String passWord = (String) myform.get("password");
        // 用户登录
        if (UserConstants.ACTION_SUBMIT.equals(actionType))
        {
            // 判断用户名和口令的正确性
            if (UserInformationManager.getInstance().checkSubmitSuccess(userId,passWord))
            {
                session.setAttribute("userId", userId);
                return mapping.findForward("success");           //①
            }
            else
            {
                request.setAttribute("check", UserConstants.CHECK_ERROR);
            }
        }
        // 判断是否已经登录
        if (session.getAttribute("userId") == null)
        {
            return mapping.findForward("init");           //②
        }
        return mapping.findForward("init");
    }
}
```

由上段代码中加黑的语句①可见, 若验证用户合法, Action 会通过 forward 返回 success, 否则返回 init (②)。

编写完 Action 后还要进行配置, 在 web.xml 中添加如下代码:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
```

```

<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
        <param-name>application</param-name>
        <param-value>LocalStrings</param-value>
    </init-param>
    <init-param>
        <param-name>config</param-name>
        <param-value>/WEB-INF/config/struts-config.xml</param-value>
    </init-param>
    <init-param>
        <param-name>config/user</param-name>
        <param-value>/WEB-INF/config/struts-config-user.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
<welcome-file-list>
    ...
</welcome-file-list>
<jsp-config>
    ...
</jsp-config>
</web-app>

```

其中 struts-config.xml 和 struts-config-user.xml 都是 Struts 的配置文件，位于项目的 WebRoot\WEB-INF\config 目录下，有关 LogonAction 的配置在文件 struts-config.xml 中，如下：

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <form-beans>
        <form-bean name="logonForm" type="org.apache.struts.action.DynaActionForm">
            <form-property name="actionType" type="java.lang.String" initial="logon"/>
            <form-property name="userId" type="java.lang.String"/>
            <form-property name="password" type="java.lang.String"/>
        </form-bean>
    </form-beans>
    <action-mappings>
        <action path="/logon" type="org.web.LogonAction" name="logonForm" scope="request">
            <forward name="init" path="/logon.jsp" />
            <forward name="success" path="/logon_success.jsp" />

```



```

        </action>
    </action-mappings>
</struts-config>

```

其中：

`<form-bean>`标签下的 `name` 属性表示 form 的名称，`type` 表示 form 的路径，此 form 调用 Struts 的 form（由 Struts 1 库提供，不需重新设计），可以在下面直接定义 form 的属性。

`<form-property>`标签用来定义各个属性的配置，`name` 表示属性名称，`type` 表示属性类型，`initial` 表示属性的默认值。

`<action>`标签下的 `path` 表示访问路径，`type` 表示实现的 class 的路径，`name` 表示 form 的名称（与`<form-bean>`标签的 `name` 属性一致），`scope` 表示访问方式。

`<forward>`标签指定返回的 JSP 页面，`name` 表示返回的名称，`path` 表示返回的 jsp 路径。这里的返回值来自 Action，由前面的分析可知，返回 `init` 表示验证未通过，系统回到初始登录页 `logon.jsp`，若成功通过验证，则会跳转到登录成功页 `logon_success.jsp`。

8.4.3 后台的验证操作

用户输入学号和口令，单击“登录”按钮后，后台的数据库操作由 `UserInformationManager` 完成，它实现了 `checkSubmitSuccess()`方法：

```

public boolean checkSubmitSuccess(String userId, String passWord)throws SQLException
{
    PreparedStatement pstmt = null;
    Connection conn = null;
    ResultSet rs = null;
    try
    {
        String sql = "SELECT * FROM " + UserConstants.TAB_XSDL+ " WHERE XH="
                                                                + userId + """;

        if (passWord != null && passWord.trim().length() > 0)
        {
            sql = sql + " AND PASSWORD='" + passWord + "'";
        }
        else
        {
            sql = sql + " AND PASSWORD IS NULL";
        }
        conn = DBConnection.getConnection();
        pstmt = conn.prepareStatement(sql);
        rs = pstmt.executeQuery();
        while (rs.next())
        {
            return true;
        }
    }
    catch (SQLException e)

```

```
{
    e.printStackTrace();
    if (conn != null){conn.close();}
}
finally
{
    if (rs != null){rs.close();}
    if (pstmt != null){pstmt.close();}
    if (conn != null){conn.close();}
}
return false;
}
```

大家可以清楚地看出代码中有 SQL 语句嵌入操作。

8.4.4 登录成功页

登录成功页 `logon_success.jsp` 的代码如下：

```
<%@page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title>学生信息系统</title>
</head>
<frameset rows="90,1*" name="frmAll">
    <frame src="menu_top.jsp" name="frmTOP" noresize>
    <frameset cols="15%,*">
        <frame src="menu_left.jsp" name="frmLEFT" noresize>
        <frame src="./user/user-list.do" name="main" noresize>
    </frameset>
</frameset>
</html>
```

页面用到了第1章所讲的框架（frameset）网页设计技术。

网页头部 `menu_top.jsp` 的代码如下：

```
<%@page language="java" contentType="text/html; charset=gb2312"%>
<html>
<body bgcolor="#8888ff">
    <div align="center">
        
        <font color="red" size="6">
            学生信息管理系统
        </font>
    </div>
</body>
</html>
```

这里使用了早已准备好的资源，即南京师范大学网站标头图片 `njnu.jpg`。

网页左部 `menu_left.jsp` 的代码如下：

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title>left 网页</title>
</head>
<body>
    <table align="center">
        <tr>
            <td>
                <a href="/user/user-list.do" target="main">学生检索<br><br>
                <a href="/user/user-insert.do" target="main">新生录入<br><br>
                <a href="logon.jsp" target="_top">退出系统<br><br>
            </td>
        </tr>
    </table>
</body>
</html>

```

左部显示三个超链接“[学生检索](#)”、“[新生录入](#)”、“[退出系统](#)”，表示系统具有的功能，其中“[学生检索](#)”链接对应的 JSP 可执行文件为 user_list.jsp，是登录后默认的载入页。用户登录后看到的页面效果如图 8.18 所示。



图 8.18 登录成功后看到的页面

8.5 学生信息检索

从图 8.18 可见，系统提供了按学号或姓名检索学生信息的功能。

8.5.1 检索页面显示

检索页面是系统默认的初始载入页，由 user_list.jsp 实现：

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="java.util.ArrayList,org.model.*,org.web.*"%>
<%@ include file="../includes/taglib.jsp"%>
<html>
<%
    UserListForm myForm = (UserListForm) pageContext.findAttribute("UserListForm");
    String userID = (String) session.getAttribute("userId");
%>
<html:form action="user-list.do" method="post">
<html:hidden property="actionType"/>
    <table border="0" cellPadding="0" cellSpacing="0" width="95%" align="center">
        <tr>
            <td height="10"></td>
        </tr>
        <tr valign="middle">
            <td>
                <input type="button" onClick="submitForm('list')" style="cursor:hand;" border="0"
                    name="search" value="检索">&nbsp;
                <input type="button" onClick="submitForm('delete')" style="cursor:hand;"
                    border="0" name="delete" value="删除">
            </td>
        </tr>
    </table>
    <tr>
        <td>
            <hr>
        </td>
    </tr>
    <tr>
        <td>
            学号: <html:text property="studentId" size="18" value=""/>&nbsp;&nbsp;&nbsp;
            姓名: <html:text property="studentName" size="18" value=""/>
        </td>
    </tr>
    <tr>
        <td>
            <hr>
        </td>
    </tr>
</table>
<table width="95%" align="center" cellSpacing="1" border="0" bordercolordark="#000000"
    bordercolorlight="#ffffff" class="list_table">
    <tr class="list_table_tr_title">
        <td noWrap width="5%">
            <input type="checkbox" name="checkAll" value="on"
                onclick="chkAllChanged(this)">
        </td>
    </tr>

```

```

        <td noWrap width="15%">
            <div align="center">学号</div>
        </td>
        <td noWrap width="15%">
            <div align="center">姓名</div>
        </td>
        <td noWrap width="10%">
            <div align="center">性别</div>
        </td>
        <td noWrap width="30%">
            <div align="center">出生日期</div>
        </td>
        <td noWrap width="30%">
            <div align="center">专业</div>
        </td>
    </tr>
</table>
<table width="95%" height="70%" align="center" cellspacing="0" border="0"
        bordercolordark="#000000" bordercolorlight="#ffffff" class="list_table">
    <tr>
        <td>
            <div id="mydiv" style="overflow:auto;width:100%;height:100%;">
                <table width="100%" bordercolor="#000000" bordercolorlight="#ffffff"
                    bordercolordark="#000000" border="0" cellspacing="1" cellpadding="0">
                    <%
                        int index = 0;
                        ArrayList studentList = (ArrayList)request.getAttribute("studentList");
                        UserBean userBean = null;
                        if(studentList!=null && studentList.size()>0)
                        {
                            int size = studentList.size();
                            for(int i=0;i<size;i++)
                            {
                                userBean = (UserBean)studentList.get(i);
                            }
                        }
                    %>
                    <tr name="trId"
                        class="<%= (index%2==0?"list_table_tr_even":"list_table_tr_odd")%>">
                        <td noWrap width="5%">
                            <div align="center">
                                <html:checkbox property="deleteRow"
                                    value="<%=userBean.getStudentId()%>" />
                            </div>
                        </td>
                        <td noWrap width="15%">
                            <%
                                if(userBean.getStudentId().equals(userID))

```

```

        {
        %>
        <div align="center">
            <a href="user-edit.do?
                actionType=refer&userId=<%=userBean.getStudentId()%>">
                <%=userBean.getStudentId()%>
            </a>
        </div>
        <%
        }
        else
        {
        %>
        <div align="center">
            <a href="user-refer.do?
                actionType=refer&userId=<%=userBean.getStudentId()%>">
                <%=userBean.getStudentId()%>
            </a>
        </div>
        <%
        }
        %>
    </td>
    <td noWrap width="15%">
        <div align="center"><%=userBean.getStudentName()%></div>
    </td>
    <td noWrap width="10%">
        <div align="center"><%=userBean.getSex()%></div>
    </td>
    <td noWrap width="30%">
        <div align="center"><%=userBean.getBirtyDate()%></div>
    </td>
    <td noWrap width="30%">
        <div align="center"><%=userBean.getSubject()%></div>
    </td>
</tr>
    <%
    index++;
    }
    %>
</table>
</div>
</td>
</tr>
</table>

```

```
</html:form>
</html>
```

8.5.2 检索 Action 模块

与登录不同，Struts 并未提供现成的 form 模块，需要用户自己定义。在 org.web 包下定义 UserListForm.java，代码如下：

```
package org.web;
import org.apache.struts.action.ActionForm;
public class UserListForm extends ActionForm {
    // action
    private String actionType;
    // 学号
    private String studentId;
    // 姓名
    private String studentName;
    // 删除用户的数组变量
    private String[] deleteRow = null;
    public String getActionType() {
        return actionType;
    }
    public void setActionType(String actionType) {
        this.actionType = actionType;
    }

    public String getStudentId()
    {
        return studentId;
    }
    public void setStudentId(String studentId)
    {
        this.studentId = studentId;
    }

    public String getStudentName()
    {
        return studentName;
    }
    public void setStudentName(String studentName)
    {
        this.studentName = studentName;
    }

    public String[] getDeleteRow()
    {
        return deleteRow;
    }
}
```

```

    }
    public void setDeleteRow(String[] deleteRow)
    {
        this.deleteRow = deleteRow;
    }
}

```

UserListAction.java 实现学生检索页面的后台操作，具体代码如下：

```

package org.web;
import java.util.ArrayList;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.model.*;
import org.util.UserConstants;
public class UserListAction extends Action
{
    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                HttpServletRequest request, HttpServletResponse response)throws Exception
    {
        HttpSession session = request.getSession();
        String userId = (String) session.getAttribute("userId");
        UserListForm myform = (UserListForm) form;
        String actionType = myform.getActionType();
        ArrayList studentList = null;
        // 人员信息 bean
        UserBean userbean = UserManager.getInstance().searchStudent(userId);
        if (UserConstants.ACTION_LIST.equals(actionType))
        {
            // 查询人员信息
            studentList = UserManager.getInstance().searchStudentList(
                myform.getStudentId(), myform.getStudentName());
            request.setAttribute("studentList", studentList);
        }
        if (UserConstants.ACTION_DELETE.equals(actionType))
        {
            // 删除人员信息
            UserManager.getInstance().delStudentValue(
                myform.getDeleteRow());
            // 查询人员信息
            studentList = UserManager.getInstance().searchStudentList(
                myform.getStudentId(), myform.getStudentName());
            request.setAttribute("studentList", studentList);
        }
        return mapping.findForward("init");
    }
}

```

检索 Action 模块的配置信息位于 struts-config-user.xml 中，如下：


```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <form-beans>
        <!-- 一览表 -->
        <form-bean name="UserListForm" type="org.web.UserListForm"/>
    </form-beans>
    <action-mappings>
        <action path="/user-list" type="org.web.UserListAction" name="UserListForm"
                                                    scope="request">
            <forward name="init" path="/user_list.jsp" />
        </action>
    </action-mappings>
</struts-config>

```

8.5.3 后台数据检索操作

可在 `UserInformationManager` 类中添加直接操作数据库的方法。

`searchStudent()` 方法根据学号检索某个学生的信息：

```

public UserBean searchStudent(String studentId) throws SQLException
{
    PreparedStatement pstmt = null;
    Connection conn = null;
    ResultSet rs = null;
    UserBean userBean = null;
    try
    {
        String sql = "SELECT * FROM " + UserConstants.TAB_XSB + ","
        + UserConstants.TAB_XSDL + " WHERE " + UserConstants.TAB_XSB + ".XH=" + studentId + """;
        conn = DBConnection.getConnection();
        pstmt = conn.prepareStatement(sql);
        rs = pstmt.executeQuery();
        boolean admin = false;
        while (rs.next())
        {
            userBean = new UserBean();
            userBean.setStudentId(rs.getString("XH"));
            userBean.setStudentName(rs.getString("XM"));
            // 获得性别
            userBean.setSex(rs.getString("XB"));
            userBean.setBirlyDate(rs.getString("CSRQ"));
            // 获得专业的名称
            userBean.setSubject(rs.getString("ZY"));
            userBean.setPassWord(rs.getString("PASSWORD"));

```

```

    }
}
catch (SQLException e){...}
finally {...}
return userBean;
}

```

而 searchStudentList()方法则能获取数据库中所有学生信息的列表:

```

public ArrayList searchStudentList(String studentId, String studentName)throws SQLException
{
    ArrayList studentList = null;
    PreparedStatement pstmt = null;
    Connection conn = null;
    ResultSet rs = null;
    UserBean userBean = null;
    try
    {
        String sql = "SELECT * FROM " + UserConstants.TAB_XSB;
        boolean hasWhere = false;
        // 添加学号检索条件
        if (studentId != null && studentId.trim().length() > 0)
        {
            sql = sql + " WHERE XH='" + studentId + "'";
            hasWhere = true;
        }
        // 添加姓名检索条件
        if (studentName != null && studentName.trim().length() > 0)
        {
            if (hasWhere)
            {
                sql = sql + " AND XM LIKE '%" + studentName + "%'";
            }
            else
            {
                sql = sql + " WHERE XM LIKE '%" + studentName + "%'";
            }
        }
        conn = DBConnection.getConnection();
        pstmt = conn.prepareStatement(sql);
        rs = pstmt.executeQuery();
        studentList = new ArrayList();
        while (rs.next())
        {
            userBean = new UserBean();
            userBean.setStudentId(rs.getString("XH"));
            userBean.setStudentName(rs.getString("XM"));
            userBean.setSex(rs.getString("XB"));
        }
    }
}

```

```

        userBean.setBirlyDate(rs.getString("CSRQ"));
        userBean.setSubject(rs.getString("ZY"));
        studentList.add(userBean);
    }
}
catch (SQLException e){...}
finally{...}
return studentList;
}

```

这里 `catch...finally{...}` 语句与之前的 `checkSubmitSuccess()` 方法一样，后面 `UserInfoManager` 类中的每个方法都有完全一样的异常捕获语句，为节省篇幅，特略去不写，请大家着重关注方法中的 SQL 语句。

8.5.4 测试检索功能

1. 按学号检索

运行程序，在图 8.18 所示的界面中输入学号“051101”，如图 8.19 所示，单击“检索”按钮。

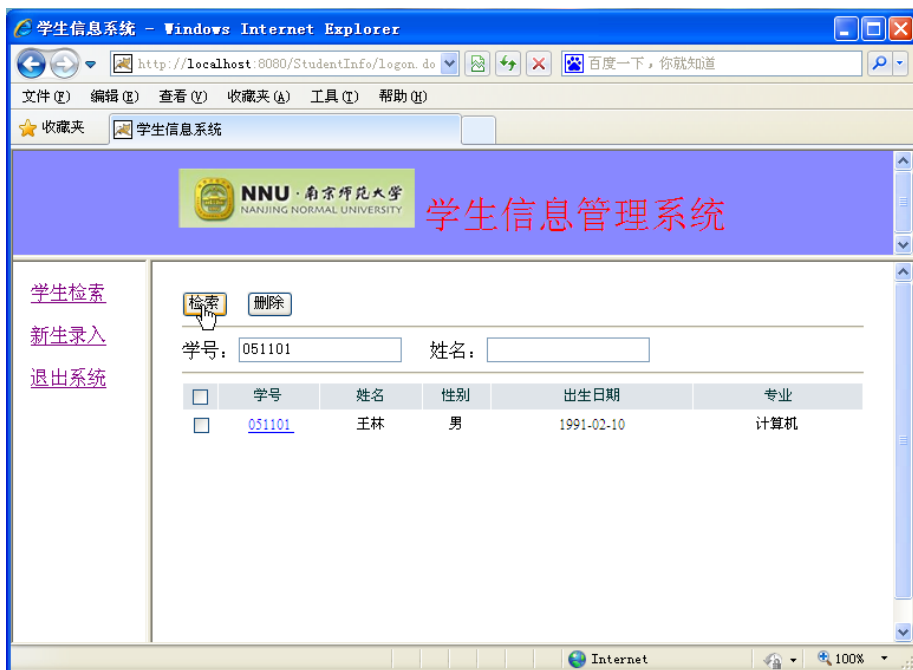


图 8.19 按学号检索

系统显示出学号为“051101”的同学王林的个人信息，包括姓名、性别、出生日期和专业。

2. 按姓名检索

在界面中的“姓名:”一栏填写“李”，单击“检索”按钮，系统会查找出数据库中所有李姓同学的信息，如图 8.20 所示。



图 8.20 按姓名检索

可见该系统还具备模糊检索的能力！

3. 学生信息一览

既然系统支持模糊检索，那么接下来我们什么都不填写，直接单击页面上的“检索”按钮，如图 8.21 所示。系统会列出所有学生的信息列表。

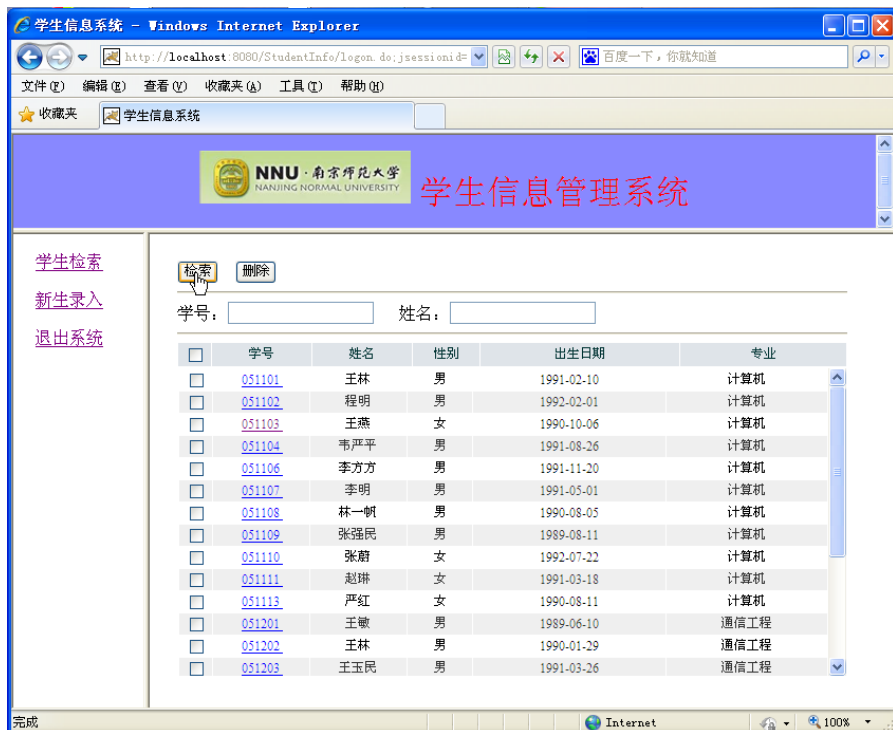


图 8.21 所有学生的信息列表

8.6 新生信息录入和删除

下面来开发新生信息录入和删除的功能。

8.6.1 新生录入页面

单击图 8.21 页面左部的“[新生录入](#)”超链接，进入学生信息录入页面，该页面由位于 WebRoot\user 目录下的 user_insert.jsp 实现：

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="org.util.*, java.util.ArrayList"%>
<%@ include file="../includes/taglib.jsp"%>
<html>
<script language="javascript">
    function submitForm(act)
    {
        // 学号
        var studentId = document.all.studentId.value;
        // 姓名
        var studentName = document.all.studentName.value;
        // 判断学号是否为空
        if(!checkNULL(studentId))
        {
            alert("学号不可以为空!");
            return;
        }
        // 判断姓名是否为空
        if(!checkNULL(studentName))
        {
            alert("姓名不可以为空!");
            return;
        }
        document.UserInsertForm.actionType.value = act;
        document.UserInsertForm.submit();
    }
</script>
<html:form action="user-insert.do" method="post">
<html:hidden property="actionType"/>
    <table border="0" cellPadding="0" cellSpacing="0" width="95%" align="center">
        <tr>
            <td height="10"></td>
        </tr>
        <tr valign="middle">
```

```

        <td>
            <input type="button" onClick="submitForm('insert');" style="cursor:hand;"
                border="0" name="search" value="插入">&nbsp;
            <input type="reset" style="cursor:hand;" border="0" name="delete" value="清空">
        </td>
    </tr>
</tr>
<tr>
    <td>
        <hr>
    </td>
</tr>
</table>
<table width="95%" align="center">
    <tr>
        <td>
            <fieldset>
                <legend><b>新生信息</b></legend>
                <table width="100%">
                    <tr>
                        <td nowrap width="15%" align="right">
                            <font color="#ff0000">*</font>&nbsp;学号:
                        </td>
                        <td width="35%" nowrap>
                            <html:text property="studentId" maxlength="25"
                                style="width:212"/>
                        </td>
                        <td nowrap width="15%" align="right">
                            <font color="#ff0000">*</font>&nbsp;姓名:
                        </td>
                        <td width="35%" nowrap>
                            <html:text property="studentName" maxlength="25"
                                style="width:212"/>
                        </td>
                        <td nowrap width="15%" align="right">
                            性别:
                        </td>
                        <td width="35%" nowrap>
                            <html:radio property="sex" value="男">
                                男</html:radio>
                            <html:radio property="sex" value="女">女</html:radio>
                        </td>
                    </tr>
                    <tr>
                        <td nowrap width="15%" align="right">
                            生日:
                        </td>

```

```

        <td width="35%" colspan="3" nowrap>
            <html:text property="birtyDate" maxlength="25"
                style="width:212"/>
        </td>
        <td noWrap width="15%" align="right">
            专业:
        </td>
        <td width="35%" nowrap>
            <select style="width:212" name="subject">
                <%
                    ArrayList subjectList = new ArrayList();
                    subjectList.add("");
                    subjectList.add("计算机");
                    subjectList.add("通信工程");
                    subjectList.add("信息网络");
                    for(int i=0;i<subjectList.size();i++)
                    {
                        String subject=subjectList.get(i).toString();
                    }
                <option value="<%=subject%>"><%=subject%></option>
                <%
                    }
                <%>
            </td>
        </tr>
    </table>
</fieldset>
</td>
</tr>
</table>
<script language="javascript">
    <%
        String checkValue = (String)request.getAttribute("check");
        if(UserConstants.CHECK_ERROR.equals(checkValue))
        {
            <%>
                alert("对不起,你插入的记录已经存在!");
            <%
                }
            <%>
        </script>
</html:form>
</html>

```

设计后的页面效果如图 8.22 所示。

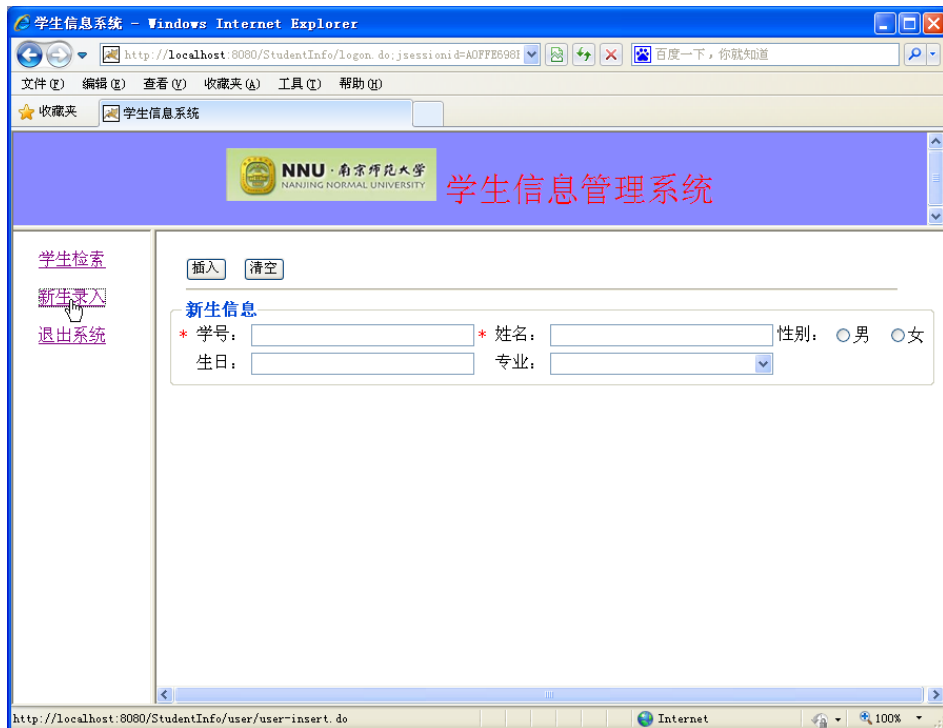


图 8.22 新生信息录入页面

在页面中的“新生信息”表单里可填写新生的学号、姓名、性别、生日和专业等信息。

8.6.2 插入 Action 模块

学生信息的录入实际上是向后台数据库中插入新记录的过程，用于插入操作的 form 模块也需要用户自己定义。

在 org.web 包下定义 UserInsertForm.java，代码如下：

```
package org.web;
import org.apache.struts.action.ActionForm;
public class UserInsertForm extends ActionForm
{
    // action
    private String actionType;
    // 学号
    private String studentId;
    // 姓名
    private String studentName;
    // 性别
    private String sex;
    // 生日
    private String birtyDate;
    // 专业
    private String subject;
```



```
public String getActionType()
{
    return actionType;
}
public void setActionType(String actionType)
{
    this.actionType = actionType;
}

public String getStudentId()
{
    return studentId;
}
public void setStudentId(String studentId)
{
    this.studentId = studentId;
}

public String getStudentName()
{
    return studentName;
}
public void setStudentName(String studentName)
{
    this.studentName = studentName;
}

public String getSex()
{
    return sex;
}
public void setSex(String sex)
{
    this.sex = sex;
}

public String getBirtyDate()
{
    return birtyDate;
}
public void setBirtyDate(String birtyDate)
{
    this.birtyDate = birtyDate;
}

public String getSubject()
```

```
{
    return subject;
}
public void setSubject(String subject)
{
    this.subject = subject;
}
}
```

UserInsertAction.java 实现插入新生记录的后台操作，具体代码如下：

```
package org.web;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.model.UserInformationManager;
import org.util.UserConstants;
public class UserInsertAction extends Action
{
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)throws Exception
    {
        HttpSession session = request.getSession();
        String userId = (String) session.getAttribute("userId");
        UserInsertForm myform = (UserInsertForm) form;
        String actionType = myform.getActionType();
        if (UserConstants.ACTION_INSERT.equals(actionType))
        {
            // 判断用户是否已经被删除
            if (userId == null || userId.trim().length() <= 0)
            {
                String url = "../logon.do?actionType=init";
                return new ActionForward(url);
            }
            else if (!UserInformationManager.getInstance().checkUserBack(userId))
            {
                String url = "../logon.do?actionType=init";
                return new ActionForward(url);
            }
            // 判断插入的数据是否已经存在
            if (UserInformationManager.getInstance().checkUserBack(myform.getStudentId()))
            {
                request.setAttribute("check", UserConstants.CHECK_ERROR);
            }
            else
            {
                // 插入数据
                UserInformationManager.getInstance().insertValue(myform);
                return mapping.findForward("success");
            }
        }
    }
}
```

```

    }
}
return mapping.findForward("init");
}
}

```

这里，在插入记录之前要先判断插入的数据是否已经存在，用到 `checkUserBack()` 方法，该方法在 `UserInfoManager` 类中，代码如下：

```

public boolean checkUserBack(String userId) throws SQLException
{
    PreparedStatement pstmt = null;
    Connection conn = null;
    ResultSet rs = null;
    try
    {
        String sql = "SELECT * FROM " + UserConstants.TAB_XSB + " WHERE XH='" + userId +
        """,

        conn = DBConnection.getConnection();
        pstmt = conn.prepareStatement(sql);
        rs = pstmt.executeQuery();
        while (rs.next())
        {
            return true;
        }
    }
    catch (SQLException e){...}
    finally{...}
    return false;
}

```

在配置文件 `struts-config-user.xml` 中对上面编写的两个 `Action` 模块进行配置，添加如下语句：

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<struts-config>
```

```
    <form-beans>
```

```
        <!-- 插入界面 -->
```

```
        <form-bean name="UserInsertForm" type="org.web.UserInsertForm"/>
```

```
    </form-beans>
```

```
    <action-mappings>
```

```
        <action path="/user-insert" type="org.web.UserInsertAction" name="UserInsertForm"
```

```
                                scope="request">
```

```
            <forward name="init" path="/user_insert.jsp" />
```

```
            <forward name="success" path="/user_list.jsp" />
```

```
        </action>
```

```
    </action-mappings>
```

```
</struts-config>
```

上段代码中黑色加阴影底纹的为需要添加的配置信息。

8.6.3 后台数据插入操作

下面讲解用代码直接在 `UserInformationManager` 类中添加插入操作的方法。
可利用 `insertValue()` 方法实现插入操作，代码如下：

```
public void insertValue(UserInsertForm form) throws SQLException
{
    PreparedStatement pstmt = null;
    Connection conn = null;
    ResultSet rs = null;
    try
    {
        // 插入 sql 语句
        String sql = "INSERT INTO " + UserConstants.TAB_XSB + "(XH,XM,XB,CSRQ,ZY,ZXF,BZ)"
            + " VALUES (" + form.getStudentId() + "," + form.getStudentName() + "," + form.getSex();
        // 设置生日
        if (form.getBirtyDate() != null && form.getBirtyDate().trim().length() > 0)
        {
            sql = sql + "," + form.getBirtyDate() + """;
        }
        else
        {
            sql = sql + ",null";
        }
        // 设置专业
        sql = sql + "," + form.getSubject() + ",null,null)";
        conn = DBConnection.getConnection();
        pstmt = conn.prepareStatement(sql);
        pstmt.execute();
    }
    catch (SQLException e){...}
    finally{...}
}
```

注意：对于插入的每个字段值，都要用“`‘`”分隔，无值的字段则写入“`null`”，所插入字段值的顺序要与数据库表的各个字段名一一对应。

8.6.4 录入新生信息

下面来测试录入功能，在录入页的表单中填写要加入的新生信息，如图 8.23 所示。

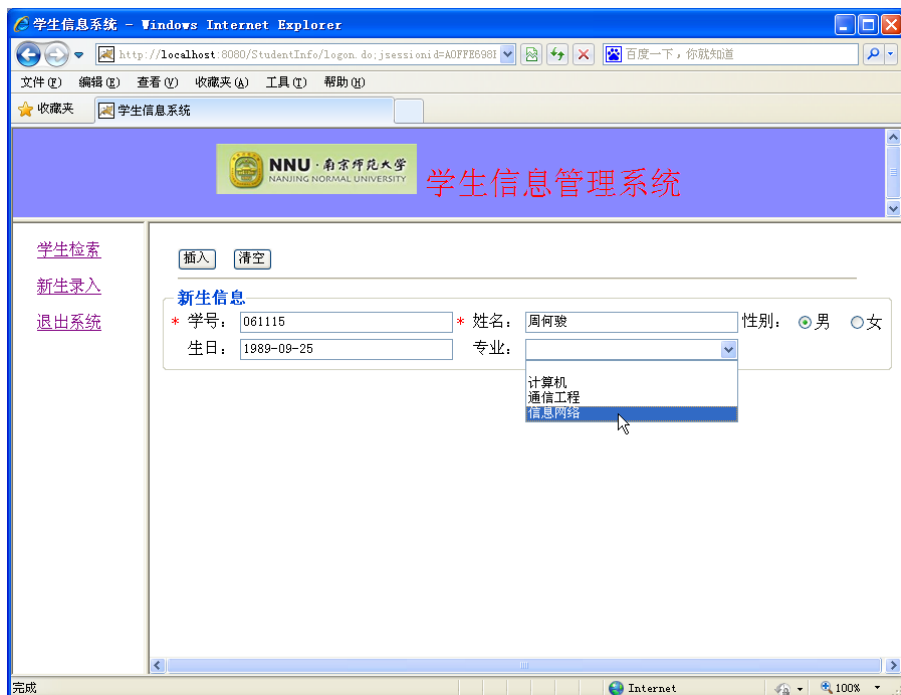


图 8.23 录入新生信息

填写完后单击“插入”按钮，跳转到学生检索页，单击“检索”按钮就会看到刚刚录入的学生信息，如图 8.24 所示（列表下方框出部分）。

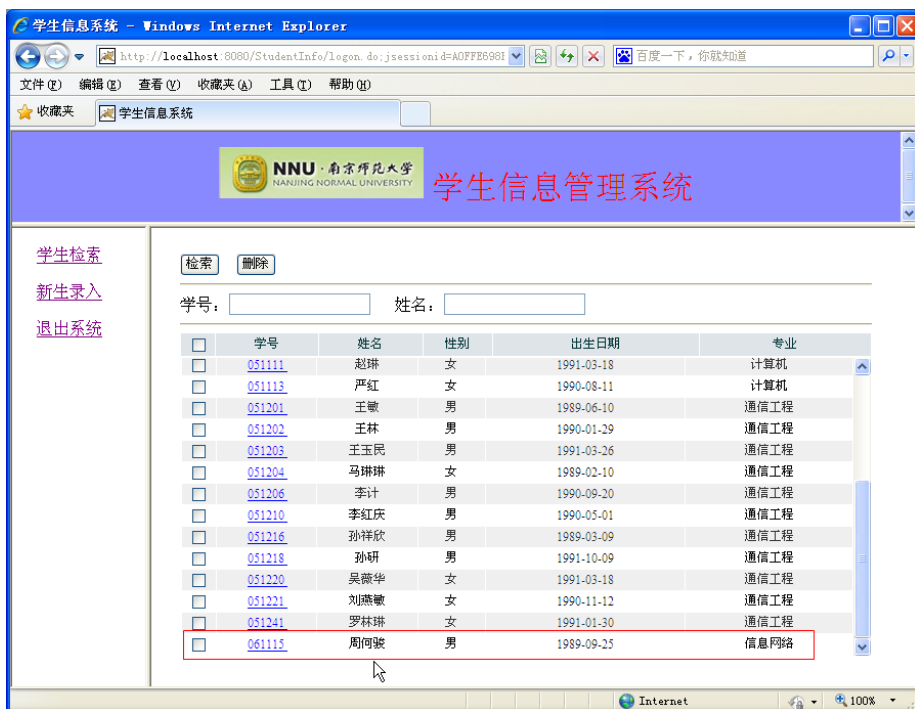


图 8.24 录入成功

用 SQL Server 2008 的 SQL Server Management Studio 直接打开数据库表,也可以看到新插入的新生记录。

8.6.5 删除学生信息

删除功能的开发很简单,只需在页面 user_list.jsp 上加入下面这段脚本:

```
<script language="javascript">
    function submitForm(act)
    {
        if(act=="delete")
        {
            var delrow = document.getElementsByName("deleteRow");
            var len = delrow.length;
            var selectNum = 0;
            for(i=0;i<len;i++)
            {
                var row = delrow[i];
                if(row.checked)
                {
                    selectNum++;
                    break;
                }
            }
            if (selectNum == 0)
            {
                alert("请至少选择一条要删除的记录");
                return;
            }
            if (!confirm("是否真的要删除?")){return;}
        }
        document.UserListForm.actionType.value = act;
        document.UserListForm.submit();
    }
    //全选操作
    function chkAllChanged(obj)
    {
        var delrow = document.getElementsByName("deleteRow");
        var lnt = delrow.length;
        if (obj.checked)
        {
            for(i=0;i<lnt;i++)
            {
                var row = delrow[i];
                row.checked = true;
            }
        }
    }
</script>
```

```

    }
    else
    {
        for(i=0;i<Int;i++)
        {
            var row = delrow[i];
            row.checked = false;
        }
    }
}
</script>

```

然后在 `UserInformationManager` 类中加入方法:

```

public void delStudentValue(String[] studentId) throws SQLException
{
    String userId = null;
    if (studentId != null && studentId.length > 0)
    {
        int len = studentId.length;
        PreparedStatement pstmt = null;
        Connection conn = null;
        try
        {
            for (int i = 0; i < len; i++)
            {
                userId = studentId[i];
                String sql = "DELETE FROM " + UserConstants.TAB_XSB + " WHERE XH="
                    + userId + """;

                conn = DBConnection.getConnection();
                pstmt = conn.prepareStatement(sql);
                pstmt.executeUpdate();
            }
        }
        catch (SQLException e){...}
        finally{...}
    }
}

```

在图 8.24 所示的学生信息列表页上, 每个学生记录前都有一个方形复选框, 想要删除哪个学生的记录, 只要选中该生记录前的复选框, 然后单击“删除”按钮即可, 如图 8.25 所示。在删除之前, 系统会弹出对话框确认。



图 8.25 删除学生记录

删除操作之后，页面会自动刷新，读者就会看到列表中已经没有那条学生的记录了。

8.7 上机练习

1. 按照本章的指导，一步一步地开发这个学生信息管理系统，重点关注 Struts 1 的工作原理以及各种 JSP 技术在其中的运用。
2. 在理解 Struts 1 原理及该系统架构方式的基础上，试着为该系统扩展新的功能，比如修改学生信息的功能。
3. 在学完 JSP 基础之后，想进一步在 Java Web 领域“深造”的读者将要学习各种成熟的框架（Struts 2/Hibernate/Spring 等），即所谓的 JavaEE 开发。Struts 2 与 Struts 1 相对用户封装了很多配置细节，更适合开发大型的 Web 应用。有意向的读者可以先行上网下载 Struts 2 框架来使用，改用 Struts 2 重新开发本章的这个案例。

《JSP 编程教程》读者意见反馈表

尊敬的读者：

感谢您购买本书。为了能为您提供更优秀的教材，请您抽出宝贵的时间，将您的意见以下表的方式（可从 <http://www.hxedu.com.cn> 下载本调查表）及时告知我们，以改进我们的服务。对采用您的意见进行修订的教材，我们将在该书的前言中进行说明并赠送您样书。

姓名：_____ 电话：_____

职业：_____ E-mail：_____

邮编：_____ 通信地址：_____

1. 您对本书的总体看法是：

☐很满意 ☐比较满意 ☐尚可 ☐不太满意 ☐不满意

2. 您对本书的结构（章节）：☐满意 ☐不满意 改进意见_____

3. 您对本书的例题：☐满意 ☐不满意 改进意见_____

4. 您对本书的习题：☐满意 ☐不满意 改进意见_____

5. 您对本书的实训：☐满意 ☐不满意 改进意见_____

6. 您对本书其他的改进意见：

7. 您感兴趣或希望增加的教材选题是：

请寄：100036 北京市万寿路 173 信箱华信大厦 1104 郝黎明 收

电话：010-88254480 E-mail: hlm@phei.com.cn



高等学校计算机教材



Access实用教程 (2007版)
ASP.NET 2.0实用教程 (第2版)
ASP.NET 3.5实用教程
AutoCAD实用教程 (第3版)
AutoCAD实用教程 (第3版) (2010中文版)
Authorware实用教程
C实用教程
C++实用教程
C++面向对象实用教程
C#实用教程
CAXA电子图板实用教程
DB2实用教程
Delphi实用教程 (第2版)
DSP实用教程
Eclipse实用教程
Illustrator实用教程
Java实用教程 (第2版)
Java EE基础实用教程

Java EE实用教程
JSP实用教程
★ JSP编程教程
MATLAB实用教程 (第3版)
MySQL实用教程
Oracle实用教程 (第3版)
Photoshop实用教程
PHP实用教程
PLC (西门子) 实用教程
PowerBuilder实用教程 (第3版)
Pro/ENGINEER实用教程
Protel实用教程
Visual Basic实用教程 (第4版)
Visual Basic.NET实用教程
Visual C++实用教程 (第4版)
Visual C#网络编程
Visual FoxPro实用教程 (第4版)
SQL Server实用教程 (第3版)
SQL Server实用教程 (第3版)
(SQL Server 2008版)
多媒体实用教程
计算机组装与维护实用教程
数据库实用教程
汇编语言实用教程
数据结构实用教程 (C语言版)



责任编辑: 郝黎明
封面设计: 孙焱津

ISBN 978-7-121-17828-3



9 787121 178283 >

定价: 39.00元